UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

IMPLEMENTACIÓN DE VEHÍCULO AUTÓNOMO EN ENTORNO SIMULADO

Trabajo Fin de Grado

AUTOR: JORGE GUTIÉRREZ MORENO

TUTOR: AGAPITO LEDEZMA ESPINO

“Si he logrado ver más lejos, ha sido porque he subido a hombros de gigantes”

Isaac Newton

Agradecimientos

En primer lugar, quiero agradecerle a mi tutor, Agapito, el haberme dado la oportunidad de desarrollar este interesante proyecto que marca el fin de una etapa muy importante de mi vida.

Con este proyecto llegan a su conclusión seis años de mi vida. Seis años en los que he crecido intelectualmente y como persona. La Universidad Carlos III ha llegado a convertirse en mi segunda casa, y despedirme de ella me trae unas sensaciones encontradas. Ahora toca abrir otra etapa de mi vida hacia lo desconocido.

Quiero darle las gracias sobre todo a mis padres, que han estado conmigo siempre que lo he necesitado, dándomelo todo sin pedir nada a cambio. Este trabajo es en parte para ellos, porque se merecen tanto como yo ver como esta etapa de mi vida tiene un final feliz y todo el esfuerzo de estos años es recompensado. Sé que seguirán estando conmigo allí donde me lleve la vida después de la universidad, como lo han estado siempre.

Muy importantes en estos años han sido mis amigos, la "familia que se elige". Gracias a Fer, Edu, Mariano, Raúl y Rober. Pueden pasar los años y quemarse etapas de la vida, pero la Agencia sigue ahí, porque primero fue una simple amistad, después se convirtió en un vínculo y poco a poco nos hemos ido convirtiendo en una familia.

Finalmente, no puedo olvidar a aquellos que me han acompañado durante estos años dentro de la universidad, compartiendo horas de estudio, jornadas interminables, partidas de mus y algún que otro trabajo. Gracias en particular a Maroto, Rubén, Andrea, Carlos, Sergio y Georgi, sin ellos estos seis años no hubiesen sido lo mismo.

Muchas gracias a todos.

Resumen

Estos últimos años ha habido un auge en el uso y desarrollo de vehículos de conducción autónoma. Cada año las empresas líderes en el sector presentan nuevos modelos, cada vez más avanzados tecnológicamente y lo que es más importante, más seguros.

La mayoría de los accidentes de tráfico están causados por fallos del conductor. Los vehículos autónomos buscan eliminar ese factor humano y convertir las carreteras del mundo en vías mucho más seguras tanto para conductores como para peatones, así como convertir la conducción en una actividad mucho más cómoda.

Este trabajo de fin de grado se basa en el desarrollo de un entorno simulado del Campus de la Escuela Politécnica Superior de la Universidad Carlos III de Madrid y del modelo de un vehículo autónomo para llevar a cabo una simulación en la que dicho vehículo circule sin problemas por el entorno del campus. Este trabajo tiene como objetivo desarrollar un controlador que funcione con el vehículo autónomo simulado y, en un futuro, con un vehículo autónomo físico propiedad de la Universidad Carlos III.

Palabras clave: vehículo autónomo, ROS, controlador, Universidad Carlos III, campus, entorno simulado, simulación, Webots.

Abstract

In the recent years there has been a boom in the use and development of autonomous driving. Every year the leading companys in the sector feature new increasingly technologically advanced models and, more importantly, safer.

Most of traffic accidents are caused by driver's errors. Autonomous vehicles pursue to eliminate that human factor and make the world's roads much safer tracks for drivers and pedestrians, as well as make driving a more confortable activity.

This final project is based on the development of a simulated environment of the Campus of the Escuela Politécnica of the Universidad Carlos III de Madrid and of the model of an autonomous vehicle with the objetive of accomplish a simulation in which the vehicle runs around the campus environment without problems. This project aims to develop a controller that works with the simulated autonomous vehicle and, in a future, with a physical autonomous vehicle owned by the Universidad Carlos III.

Key terms: autonomous vehicle, ROS, controller, Universidad Carlos III, campus, simulated environment, simulation, Webots.

Índice

Índice de Figuras

Índice de Tablas

CAPÍTULO 1
Introducción

1. Introducción

Este año 2016 se cumplen 130 años del comienzo de la era del automóvil. En 1886 el ingeniero Karl Friederich Benz patentó el primer vehículo automotor de combustión interna de la historia, al que puso el nombre de *Benz Patent-Motorwagen*.

Unos años más tarde, en 1908, el empresario Henry Ford, padre de las cadenas de producción en masa modernas, comenzó la producción de automóviles en serie en una cadena de montaje, otro hito en la historia del automóvil que permitió que los vehículos de combustión interna se convirtieran poco a poco en el medio de transporte popular que son hoy en día.

1.1. Motivación

En los últimos 130 años, el automóvil ha evolucionado a pasos agigantados. Esta evolución no se ha centrado únicamente en conseguir vehículos más rápidos y potentes, si no también vehículos más seguros.

Con la extensión del uso del automóvil, también apareció una de las mayores tragedias de la era moderna, las muertes por accidente de tráfico. Según un informe realizado por la OMS (Organización Mundial de la Salud) en 2013 sobre seguridad vial [1], cada año fallecen unos 1,25 millones de personas en accidentes de tráfico y entre 20 y 50 millones resultan heridas. Estos datos son alarmantes y explican cuál es la tendencia en el diseño de automóviles en la actualidad; el desarrollo de sistemas de seguridad y de asistencia al conductor.

Estos sistemas de asistencia al conductor, también conocidos como Sistemas Avanzados de Asistencia a la Conducción o ADAS (por las siglas en Inglés de *Advanced Driving Assistance Systems*), tienen como objetivo eliminar, o al menos reducir, la influencia de los errores humanos durante la conducción, que son la causa de la mayor parte de los accidentes de tráfico. Pero la tendencia actual no son únicamente mejores sistemas de asistencia, también se están desarrollando automóviles completamente autónomos, capaces de circular sin intervención alguna del conductor, eliminando así por completo el factor humano en la conducción, con las ventajas e inconvenientes que eso conlleva.

El objetivo de este trabajo es contribuir al desarrollo de vehículos autónomos, con el fin de hacer de la conducción una actividad más segura.

1.2. Descripción del Problema

Implementar un controlador para un vehículo, es decir, un programa informático encargado de interaccionar con el vehículo para dirigir su funcionamiento, tiene un problema principal, y es que requiere un elevado gasto de recursos, tanto temporales como económicos a la hora de realizar las pruebas para comprobar el correcto funcionamiento de dicho controlador. Además de estos problemas de recursos, se encuentran los problemas de seguridad, ya que por la naturaleza de las pruebas que hay que realizar con el vehículo, éstas no se pueden realizar directamente en la vía pública debido al riesgo que suponen tanto para el propio vehículo como para el personal que realiza las pruebas, además del resto de usuarios de la vía donde se realicen.

Por tanto, para realizar estas pruebas se requiere simular digitalmente tanto el vehículo como el entorno donde va a operar éste para poder realizarlas de forma segura, controlada y repetible.

1.3. Alcance del proyecto

El objetivo principal de este trabajo fin de grado es desarrollar un controlador bajo la infraestructura ROS (*Robot Operating System*) [2] para un vehículo autónomo en un entorno simulado que recreará el campus de la Escuela Politécnica de la Universidad Carlos III de Madrid.

Los objetivos específicos del proyecto son:

- Desarrollar el entorno simulado del campus de la Escuela Politécnica de la Universidad Carlos III de Madrid.
- Desarrollar un modelo simulado del vehículo autónomo que se va a utilizar.
- Implementar un controlador que opere el vehículo autónomo en el entorno del campus simulado en el lenguaje de programación C++.
- Implementar y validar el controlador anterior bajo la infraestructura ROS.

1.4. Organización de la memoria

A continuación, se describe la organización de este documento, exponiendo brevemente cual es el contenido de cada capítulo.

En el Capítulo 1 se realiza una introducción al proyecto sobre el que trata esta memoria. Esta introducción comprende la motivación para la realización del proyecto, la descripción del problema que se plantea, el alcance que tiene el proyecto y, por último, la organización del propio documento correspondiente a la memoria del proyecto.

En el Capítulo 2 se expone un análisis del estado del arte tanto en lo que se refiere a vehículos autónomo, como a la robótica y los simuladores de conducción.

En el Capítulo 3 se detalla el análisis y diseño del sistema que se ha implementado. Este análisis incluye la descripción de las normas y restricciones bajo las que se realiza el proyecto, su entorno operacional, y la especificación de sus requisitos, tanto funcionales como no funcionales, y casos de uso.

En el Capítulo 4 se muestra el proceso y resultados de los elementos que forman la simulación. Los elementos simulados que se muestran son el campus de la Escuela Politécnica Superior de la Universidad Carlos III de Madrid y el vehículo autónomo.

En el Capítulo 5 se realiza una descripción de la implementación del controlador desarrollado en este proyecto. Esta descripción incluye la implementación en C++ del controlador, así como su versión en ROS. Finalmente se muestran los resultados de la experimentación llevada a cabo con el controlador implementado.

En el Capítulo 6 se define la planificación del trabajo y los presupuestos del proyecto. Se explica la metodología seguida para el desarrollo del proyecto, la planificación de trabajo elaborada y los presupuestos del proyecto desglosados en coste de personal, coste material y coste total de proyecto.

En el Capítulo 7 se exponen las conclusiones, tanto técnicas como personales, del trabajo realizado, así como las futuras líneas de trabajo que se pueden seguir a partir del trabajo realizado y las posibles mejoras que se pueden realizar sobre el mismo.

Como parte final del documento se incluyen la bibliografía y referencias, así como los anexos.

CAPÍTULO 2
Estado del arte

2. Estado del arte

Antes de comenzar a explicar en detalle el contenido del proyecto, es necesario ponerlo en contexto mediante su estado del arte. En este capítulo se expondrá una introducción a la situación actual de los vehículos autónomos, a la robótica y los paradigmas con los que trabaja y al uso de simuladores tanto para conducción como para robótica.

2.1. Vehículos Autónomos

Un coche autónomo (también llamado coche robótico o coche auto-conducido) es un vehículo capaz de analizar el medio en el que se encuentra y navegar por él sin que sea necesaria la operación mecánica del conductor.

Los coches autónomos detectan sus alrededores mediante el uso de radares, láser, sistema de posicionamiento global y visión computarizada. La información recogida por los sensores es interpretada por sistemas avanzados de control para identificar rutas de navegación apropiadas, así como obstáculos y señales de tráfico. [3]

Los primeros coches autosuficientes (por tanto, verdaderamente autónomos) aparecieron en la década de 1980 con los proyectos Navlab y ALV de *la Carnegie Mellon University* (Pittsburgh, Pennsylvania) en 1984 y el proyecto EUREKA Prometheus de Mercedes-Benz y la Universidad de Múnich en 1987. Desde entonces, numerosas compañías y grupos de investigación han trabajado en el desarrollo de prototipos de vehículos autónomos, siendo las más reconocidas en la actualidad las empresas Google y Tesla Motors.

2.1.1. Clasificación

Actualmente existen dos sistemas propuestos de clasificación para los coches autónomos: el de la *National Highway Traffic Safety Administration* (NHTSA) en Estados Unidos y el de la Sociedad de Ingenieros Automotrices (SAE). Ambas clasificaciones son muy parecidas, por lo que solo se va a exponer la clasificación propuesta por la SAE al tratarse de un organismo internacional de estandarización.

2.1.1.1. Estándar SAE J3016

El estándar J3016 [4] publicado por SAE International tiene seis objetivos principales:

1. Identificar seis niveles de conducción automática, desde “sin automatización” a “automatización completa”.
2. Establecer una base de niveles y definiciones sobre aspectos funcionales de la tecnología
3. Describir las diferencias de forma categórica para una progresión gradual a través de los distintos niveles de automatización
4. Ser consistente con las actuales prácticas de la industria
5. Aclarar las posibles confusiones y ser útil en numerosas disciplinas (ingeniería, legal, medios, y discurso público).

6. Educar a una comunidad más amplia, aclarando para cada nivel cual es el papel (si lo hay) de los conductores en la realización de la tarea de conducción dinámica mientras se usa un sistema de conducción automática.

Los seis niveles que determina este estándar se basan en la cantidad de intervención y atención requerida por el conductor del vehículo autónomo y son los siguientes:

- **Nivel 0: Sin automatización.** El sistema automático no tiene control sobre el vehículo, pero puede emitir alertas al conductor.
- **Nivel 1: Ayuda al conductor.** Permite la ejecución de un modo específico de conducción por un sistema automatizado que controla la dirección o la aceleración/deceleración durante la conducción El conductor debe estar listo para tomar el control del vehículo en cualquier momento.
- **Nivel 2: Automatización Parcial.** El conductor está obligado a detectar obstáculos y eventos, y a responder si el sistema automático no es capaz de responder de forma adecuada. El sistema automático se encarga de la aceleración, el frenado y la dirección. Dicho sistema puede ser desactivado inmediatamente, siendo el control del vehículo tomado por el conductor.
- **Nivel 3: Automatización condicional.** Dentro de limitados entornos conocidos (como autopistas) el conductor puede dejar de prestar atención de las tareas de conducción de forma segura, pero tiene que estar preparado por si el sistema requiere su intervención.
- **Nivel 4: Alta automatización**. El sistema automático puede controlar el vehículo en casi todos los ambientes, como el mal tiempo. El conductor debe activar el sistema automático solo cuando es seguro hacerlo. Una vez activado, no se requiere atención alguna por parte del conductor.
- **Nivel 5: Automatización completa.** No se requiere atención humana más allá de establecer el destino e iniciar el sistema. El sistema automático puede conducir a cualquier localización donde sea legal conducir.

A continuación (ver Figura 1) se muestran los niveles definidos anteriormente siguiendo un orden de menor a mayor según las tareas de conducción desempeñadas por el sistema automático de conducción.

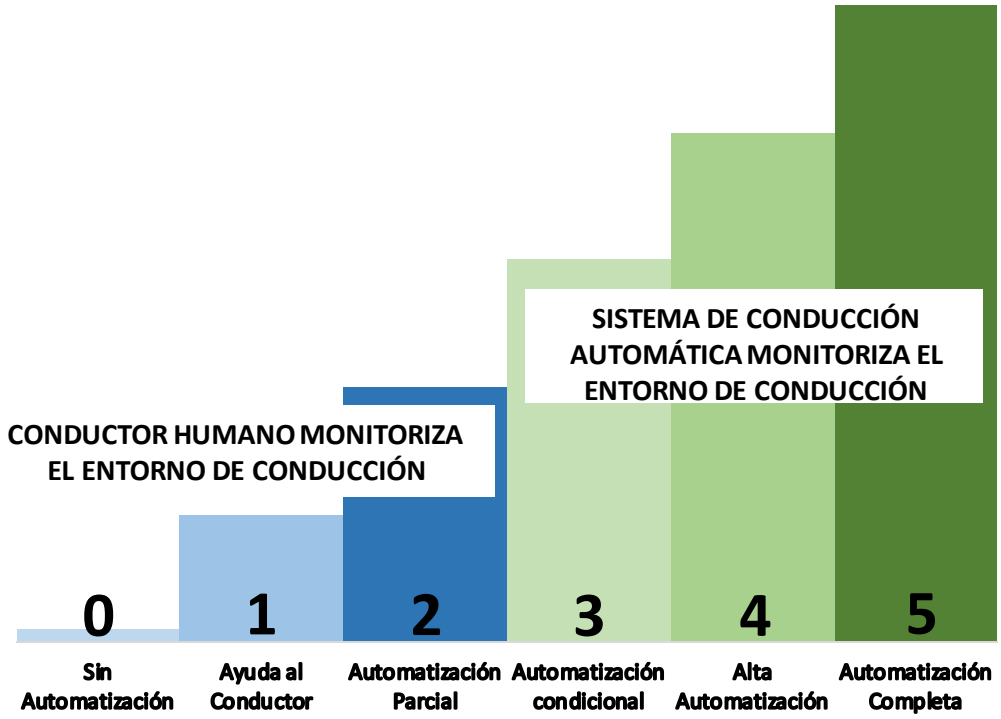


Figura 1: Niveles de conducción automática

La Tabla 1 muestra las características correspondientes de cada uno de los niveles descritos anteriormente.

Nivel SAE	Nombre	Uso de Dirección y Aceleración/Deceleración	Monitorización del Entorno de Conducción	Ejecución de Tareas de Conducción Dinámica	Capacidad del Sistema (Modos de Conducción
0	**Sin Automatización**	Conductor Humano	Conductor Humano	Conductor Humano	N/A
1	**Ayuda al Conductor**	Conductor Humano y Sistema	Conductor Humano	Conductor Humano	Algunos Modos de Conducción
2	**Automatización Parcial**	**Sistema**	Conductor Humano	Conductor Humano	Algunos Modos de Conducción
3	**Automatización Condicional**	Sistema	**Sistema**	Conductor Humano	Algunos Modos de Conducción
4	**Alta Automatización**	Sistema	Sistema	**Sistema**	Algunos Modos de Conducción
5	**Automatización Completa**	Sistema	Sistema	Sistema	**Todos los modos de conducción**

Tabla 1: Características niveles SAE

2.1.2. Sistema ADAS

Los Sistemas Avanzados de Asistencia al Conductor (*Advanced Driver Assistance Systems*, ADAS) son sistemas encargados de ayudar al conductor en el proceso de conducción y forman parte de las funcionalidades de los coches autónomos [5]. Algunos de los principales sistemas de avanzados de asistencia al conductor son:

- **Sistema de aviso de colisión frontal (ADAS de Nivel 0, No Automatizado):** el sistema de aviso de colisión frontal (FCW por sus siglas en Inglés) [6], alerta al conductor cuando el vehículo está a punto de colisionar con otro vehículo que se encuentra a una distancia específica por delante del vehículo. El tipo de avisos que usa el vehículo pueden variar entre modelos, unos usan una luz intermitente, mientras otros usan una alarma de sonido o vibración.

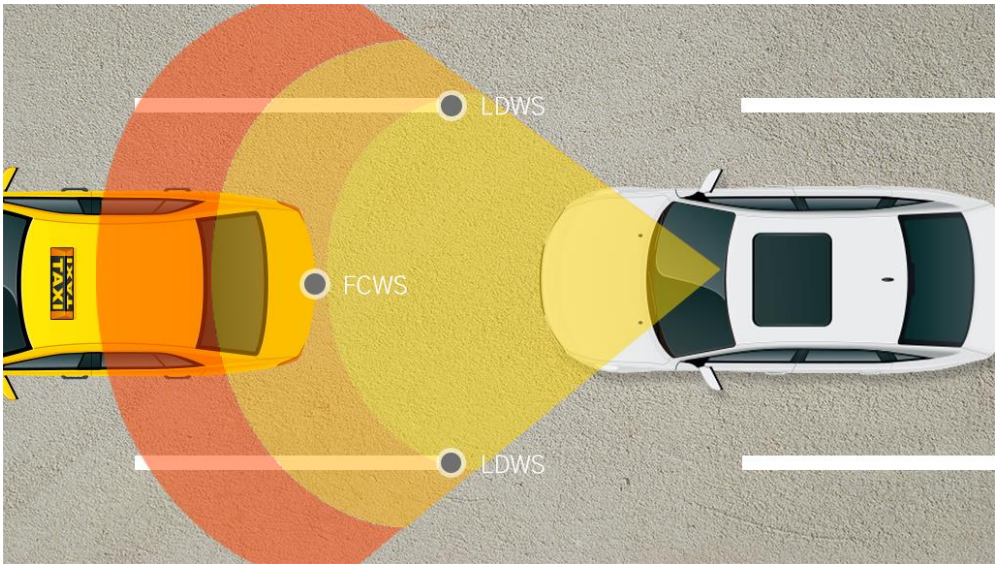


Figura 2: Ejemplo de sistema de aviso de colisión frontal [7]

- **Sistema de aviso de cambio de carril (ADAS de Nivel 0, No Automatizado):** se trata de un mecanismo diseñado para avisar al conductor cuando el vehículo comienza a moverse fuera de su carril sin estar activa la intermitencia en la dirección del movimiento [8].

Figura 3: Ejemplo de sistema de aviso de cambio de carril [9]

- **Control de Crucero Adaptativo (ADAS de Nivel 1, Asistencia al Conductor):** este sistema se encarga de ajustar automáticamente la velocidad del vehículo para mantener una distancia segura con los vehículos que tiene delante mediante el uso de sensores laser o radares [10].

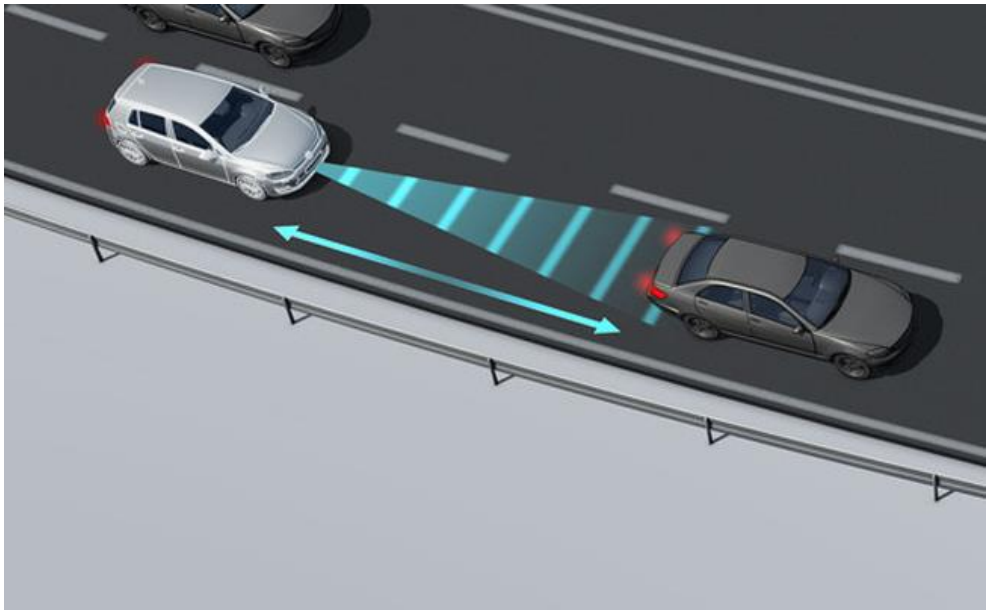


Figura 4: Ejemplo de control de crucero adaptativo [11]

- **Frenado de emergencia autónomo (ADAS de Nivel 1, Asistencia al Conductor):** este sistema se emplea para detectar una colisión inminente y realizar un frenado emergencia para evitar dicha colisión sin necesidad de intervención del conductor [12].

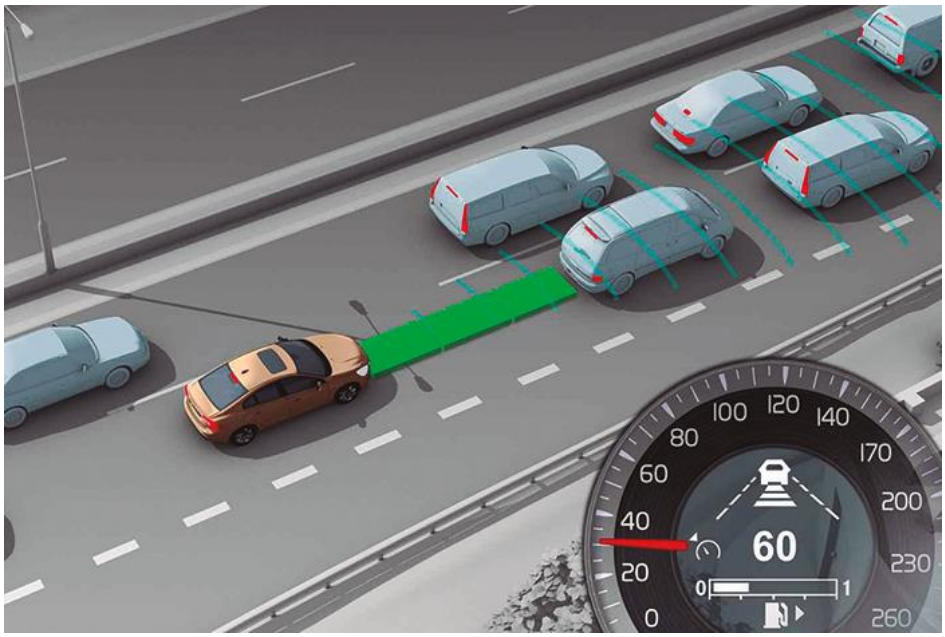


Figura 5: Ejemplo de aparcamiento automático [12]

- **Aparcamiento automático (ADAS de Nivel 2, Automatización Parcial):** este sistema autónomo de maniobrabilidad del vehículo se encarga de mover el vehículo desde la calzada hasta una plaza de aparcamiento para realizar un aparcamiento en paralelo, en batería o perpendicular [13].

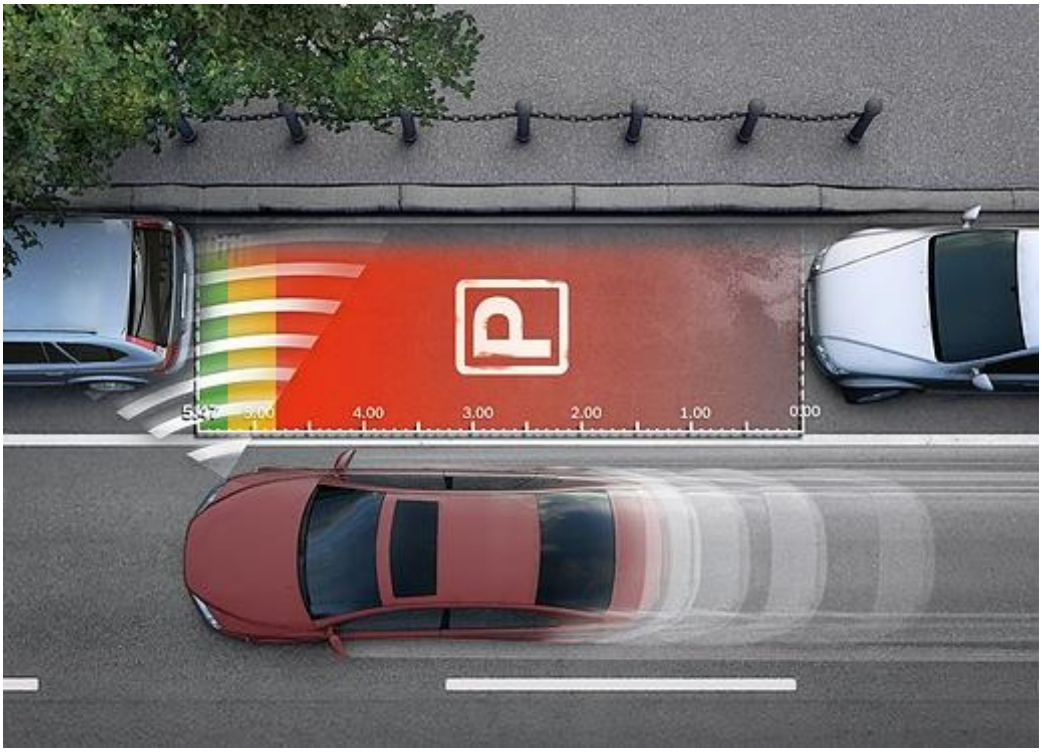


Figura 6: Ejemplo de aparcamiento automático [14]

- **Asistente en atascos (ADAS de Nivel 2, Automatización Parcial):** este sistema conocido como *Traffic Jam Assist,* se encarga de asistir al conductor cuando este se encuentra en un atasco de tráfico. Cuando el conductor activa este sistema, el vehículo detiene y reanuda la marcha de forma automática, manteniendo en todo momento la distancia de seguridad, sin necesidad de que el conductor intervenga [15].

Figura 7: Ejemplo de asistente en atascos [16]

2.2. Robótica

Como se ha comentado en el apartado anterior, los vehículos autónomos son también llamados "robóticos". Esto quiere decir que los vehículos autónomos forman parte del mundo de la robótica y por lo tanto en este apartado se realiza una introducción a esta ciencia.

2.2.1. Introducción a la Robótica

La robótica es la rama de la ingeniería mecánica, eléctrica e informática que se encarga del diseño, construcción, operación y aplicación de robots, así como de los sistemas para su control, uso de sensores y procesado de la información [17].

Estas tecnologías trabajan con máquinas automáticas que pueden sustituir a los humanos en entornos peligrosos o procesos de manufactura, así como asemejarse a humanos en apariencia, comportamiento y cognición.

A lo largo de la historia, ha sido frecuentemente asumido que un día los robots serán capaces de imitar el comportamiento humano y gestionar tareas de una forma parecida a la humana. La realidad es que, hoy en día, el campo de la robótica está en una rápida expansión y, mientras continúan los avances tecnológicos, se construyen nuevos robots con diferentes usos prácticos ya sean para uso doméstico, comercial o militar.

El vehículo autónomo que se utiliza en este proyecto es considerado un robot, ya que es una máquina que realiza operaciones de conducción antes reservadas únicamente a los humanos y que, por tanto, se encuentran bajo la rama de la robótica.

En la Figura 8 se describen los elementos básicos que forman un robot: un sistema de control, un sistema electromecánico, un conjunto de sensores con los que el robot adquiere la información del entorno en el que se encuentra y una serie de actuadores, encargados de cambiar su estado en función de la situación en la que se encuentre.

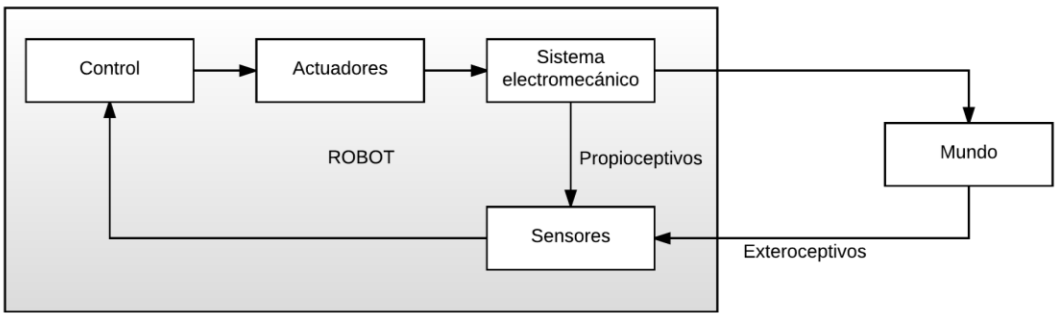


Figura 8: Componentes de un robot

2.2.2. Arquitecturas de control inteligente de robots móviles

Las arquitecturas de control detalladas a lo largo de este apartado, están recogidas en un análisis de arquitecturas de control distribuido [18]. Antes de detallar las arquitecturas de control inteligente de robots móviles, vamos a definir lo que es una arquitectura de control.

2.2.2.1. Arquitectura de control

En informática la arquitectura de sistema puede definirse como el conjunto de reglas y métodos que describen tanto la funcionalidad como la organización y la implementación de un sistema [19]. Si se aplica un punto de vista más práctico, la arquitectura de un sistema puede entenderse como un concepto abstracto que puede utilizarse para describir un sistema.

Los componentes más comunes de una arquitectura de control se pueden ver en la Figura 9. Estos componentes dependen del tipo de control que se realice, pero suelen ser los encargados de adquirir y procesar la información, así como de la toma de decisiones sobre las acciones que debe realizar el sistema y la gestión de sus tareas.

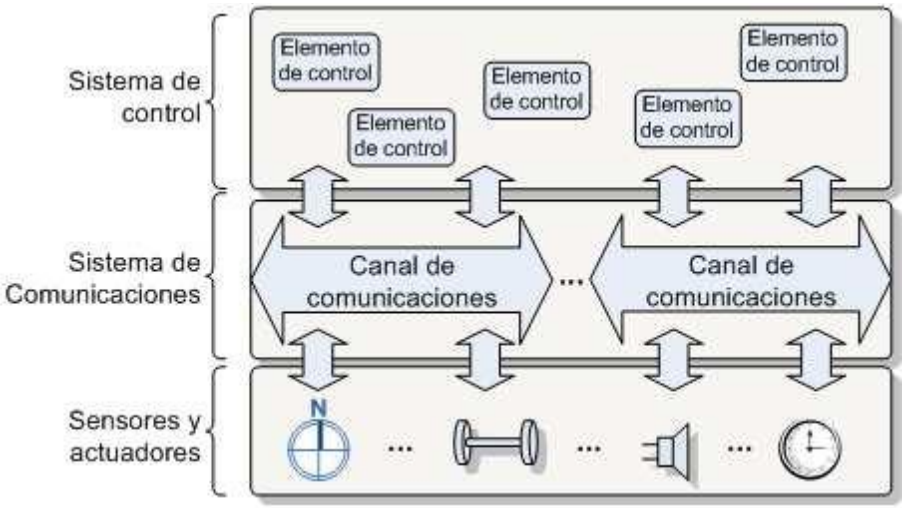


Figura 9: Componentes de una arquitectura de control distribuido [18]

Existen multitud de maneras de definir el concepto de control del sistema, pero básicamente dice que el control del sistema es el proceso por el que se un sistema se guía para cumplir una serie de objetivos establecidos, expresado como una serie de resultados enmarcados dentro de unos parámetros establecidos. Hay que tener en cuenta que, si el objetivo del control es en un corto espacio y tiempo, como por ejemplo que un robot móvil esquive un obstáculo próximo, se habla de control reactivo. Según se va ampliando el ámbito espaciotemporal de control del sistema, se comienza a hablar de control deliberativo. Es destacable que los componentes de una arquitectura varían de forma considerable entre el ámbito reactivo y el deliberativo.

2.2.2.2. Paradigmas de control

Normalmente cada sistema tiene una arquitectura particular, pero en general, éstas se desarrollan según dos paradigmas: el paradigma de control reactivo y el paradigma de control deliberativo. Existe también un tercer paradigma que combina estos dos, llamado paradigma de control híbrido, que actualmente cuenta con más aceptación que los anteriores. A continuación, se detallan estos paradigmas.

Paradigma reactivo

El control dentro del paradigma reactivo se basa en el principio de acción-reacción (ver Figura 10). La base de este paradigma es intentar imitar los comportamientos básicos de los animales [20]. Son sistemas basados en reacciones, es decir, no se cuenta con planes elaborados o metas, si no que se simulan comportamientos como pueden ser los de huir, esquivar obstáculos, etc. Las acciones que se llevan a cabo en este paradigma son provocadas por estímulos a través de los sensores del robot, no existe dependencia de una capa inteligente para que el comportamiento del robot lo parezca, el funcionamiento del sistema se basa en una dualidad Sentir-Actuar.

Gracias a los procesos basados en la dualidad Sentir-Actuar, se construyen lo que se denomina patrones de comportamiento. Estos patrones de comportamiento recogen la información captada por los sensores y se encargan de generar las señales correspondientes en los actuadores. Dentro de un sistema pueden estar ejecutándose de forma concurrente diferentes patrones de comportamiento ya que las acciones que provocan uno de estos comportamientos son independientes de las que generan el resto de comportamientos. El resultado de esta concurrencia es que algunos patrones de comportamiento son eclipsados por otros, generando un comportamiento que no es básico, si no que se trata de una combinación de comportamientos que se denomina comportamiento emergente.

Figura 10: Esquema del paradigma de control reactivo [18]

Los sistemas reactivos no dependen de algoritmo alguno, por lo que en estos sistemas poseen una capacidad de predicción temporal que permite que en el mundo real se puedan cumplir restricciones. Además, como no existe ningún tipo de representación interna en los sistemas reactivos, no es necesario realizar ningún tipo de planificación o localización, por lo que los razonamientos complejos son innecesarios, así como las complejas estructuras de datos que se usan para representar el entorno.

Arquitecturas reactivas

Existen diversas arquitecturas que utilizan un paradigma reactivo, siendo las más reconocidas la arquitectura de subsunción (en Inglés, *subsumption*) y los vehículos de Braitemberg.

Vehículos de Braitemberg

En los vehículos de navegación de Braitemberg existe una conexión directa, o al menos basada en combinaciones sencillas, entre sensores y actuadores [21] .En la Figura 11, se puede observar un ejemplo de dichas conexiones.

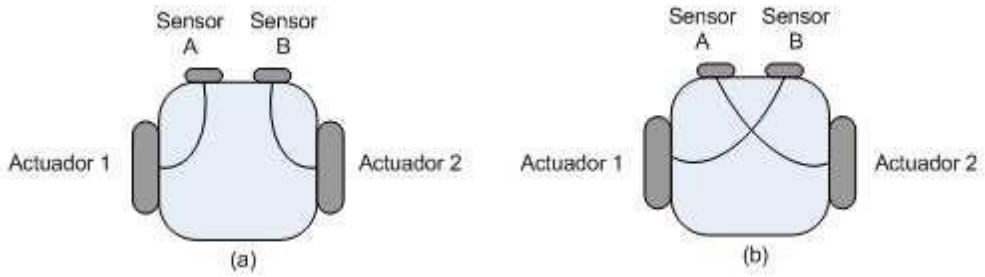


Figura 11: Ejemplo de Vehículos de Braitemberg [18]

Los vehículos de Braitemberg son una representación muy básica de los principios de la navegación reactiva. Desde un punto de vista arquitectónico (aunque hay que tener en cuenta que no constituyen una arquitectura en si mismos), son una cota inferior ya que representan unicamente la información y comunicaciones más básicas de la implementación de una aqruitectura reactiva. Algo a tener en cuenta en estos vehículos es que es prácticametne imposible predecir el comportamiento que van a tener, pese a que las conexiones entre sensores y actuadores son básicas. Sim embargo, es posible ajustar el comportamiento de estos vehículos mediante la variación de sus conexiones.

Subsumption

Esta arquitectura, que es no simbólica y no verbal, es usada como alternativa a las arquitecturas tradicionales dentro de la Inteligencia Artificial (ver Figura 12.a). La arquitectura de subsunción es usada también para ampliar otras arquitecturas más complejas haciendo uso de sus módulos básicos. La arquitectura se describe inicialmente en Brooks, 1986 [20] y surge como una aproximación de procesamiento paralelo (ver Figura 12.b) alternativa a la aproximación secuencial de la inteligencia artificial.

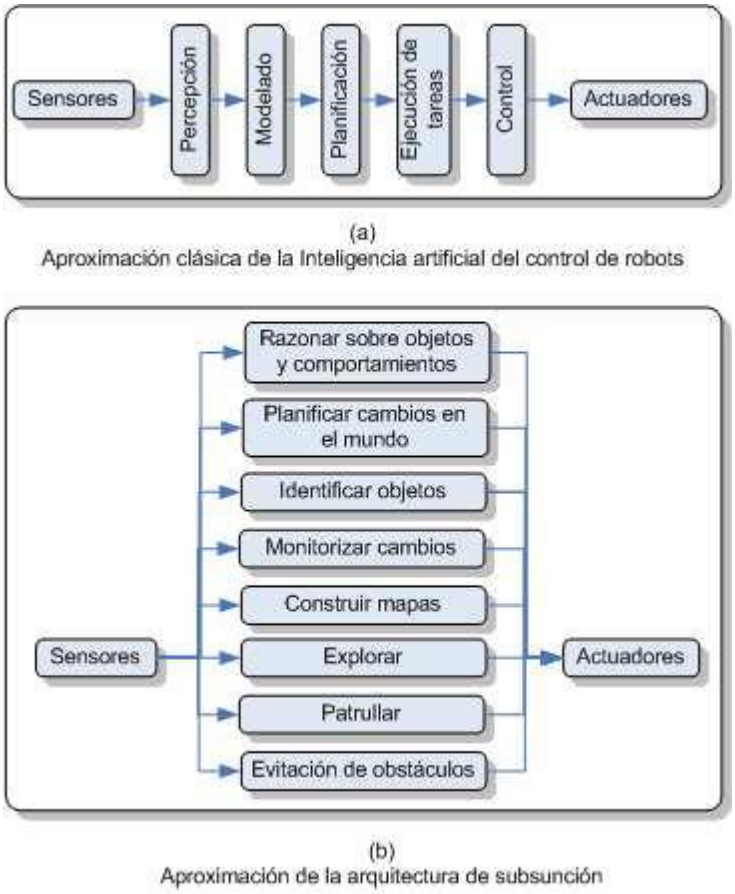


Figura 12: Aproximación clásica de la Inteligencia Artificial (a) y aproximación de la subsunción (b) [18]

En la arquitectura de subsunción los sensores son las entradas de información para cada una de las capas. Cada capa procesa dicha información y ofrece unos resultados que a su vez tendrán un efecto sobre los actuadores. La implementación de cada una de estas capas se puede hacer de maneras diferentes, por ejemplo, usando máquinas de estado finito o redes neuronales.

Paradigma deliberativo

El control deliberativo [22] toma decisiones sobre las acciones a realizar en base al razonamiento (ver Figura 13).

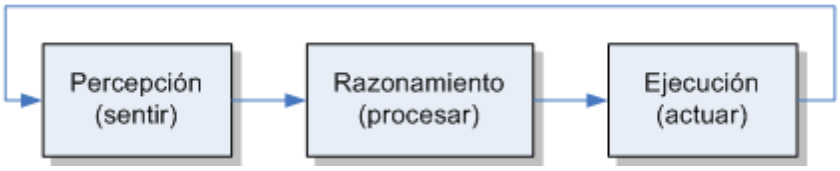


Figura 13: Esquema del paradigma de control deliberativo [18]

Que se realicen acciones de razonamiento implica que se disponga de una representación interna del entorno y de una serie de algoritmos que se encarguen de que se ejecuten las acciones decididas en base a esa representación. No obstante, la cantidad de información que es necesario almacenar es demasiado extensa, por lo que no es viable su almacenamiento [23]. Por otra parte, el retardo existente entre la percepción del entorno que realiza el robot mediante sus sensores y la ejecución de las acciones pertinentes debido a que tiene que llevarse a cabo el proceso de razonamiento, puede causar que los datos percibidos no coincidan con los datos reales en el momento en el que se produce la ejecución de las acciones porque el entorno del robot ha cambiado durante ese retardo.

Arquitecturas deliberativas

En un robot autónomo, no tiene sentido establecer una arquitectura puramente deliberativa. Se pueden encontrar aproximaciones deliberativas en robots en los que cada uno de los pasos que se llevan a cabo son extremadamente importantes y han de realizarse con extrema precisión y cuidado, como pueden ser los vehículos de exploración espacial. Dentro de las arquitecturas deliberativas existen dos modelos: el modelo secuencial y el modelo paralelo.

Modelo secuencial: JPL

El objetivo con el que se desarrolló la arquitectura JPL (*Jet Propulsion Laboratory Architecture*) fue el de dotar de una autonomía parcial a los vehículos de exploración planetaria. Esta arquitectura está compuesta por cuatro módulos principales (ver Figura 14): el módulo de percepción, el Planificador de Caminos, el Planificador y Monitor de Ejecución y el sistema Ejecutor.

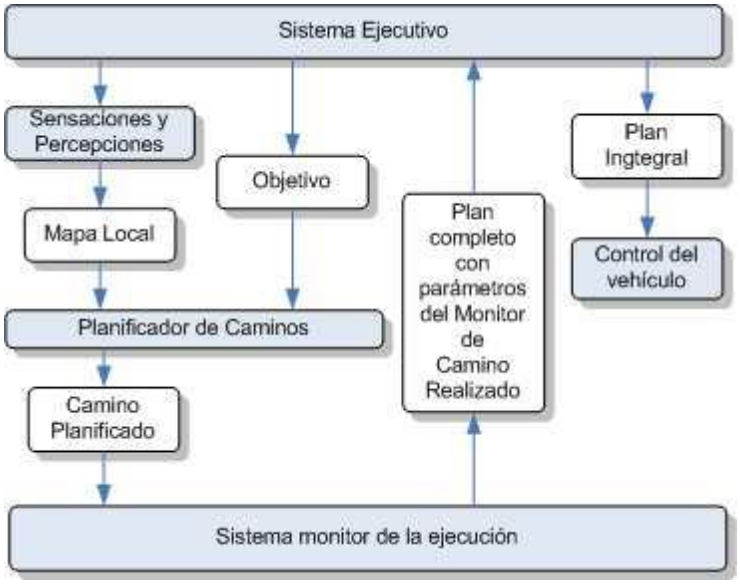


Figura 14: Esquema de la arquitectura JPL [18]

El paradigma sobre el que se fundamenta esta arquitectura es el paradigma secuencial denominado SMPA (*Sense-Model-Plan-Act*), cuyo funcionamiento se basa en ejecutar maniobras predefinidas cuando el robot detecta situaciones de peligro.

Modelo paralelo: NASREM

NASREM es el modelo estándar de referencia de la NASA (*NASA Standard REference Model*). El modelo es creado como una arquitectura de un sistema RCS (*Real-Time Control System*) [24] y es el estándar que usa la NASA como referencia para sus robots. Para implementar esta arquitectura como un sistema real, los elementos inteligentes que la forman se integran mediante módulos de computación interconectados mediante redes y jerarquías de interconexión. El modelo interno del entono real del robot es proporcionado mediante el procesamiento de los datos que proporcionan sus sensores a un sistema de procesamiento. Para llevar a cabo los comportamientos de control del robot, se generan unas acciones sobre los actuadores del propio robot que deben encontrarse dentro del contexto del modelo del mundo percibido. El modelo NASREM está formado por seis elementos básicos: actuadores, sensores, procesamiento sensorial, modelo del mundo, generador de comportamientos, estimador de valores.

Este modelo está formado por seis capas, organizadas en tres columnas: procesamiento sensorial, modelador del mundo y descomposición de tareas (ver Figura 15):

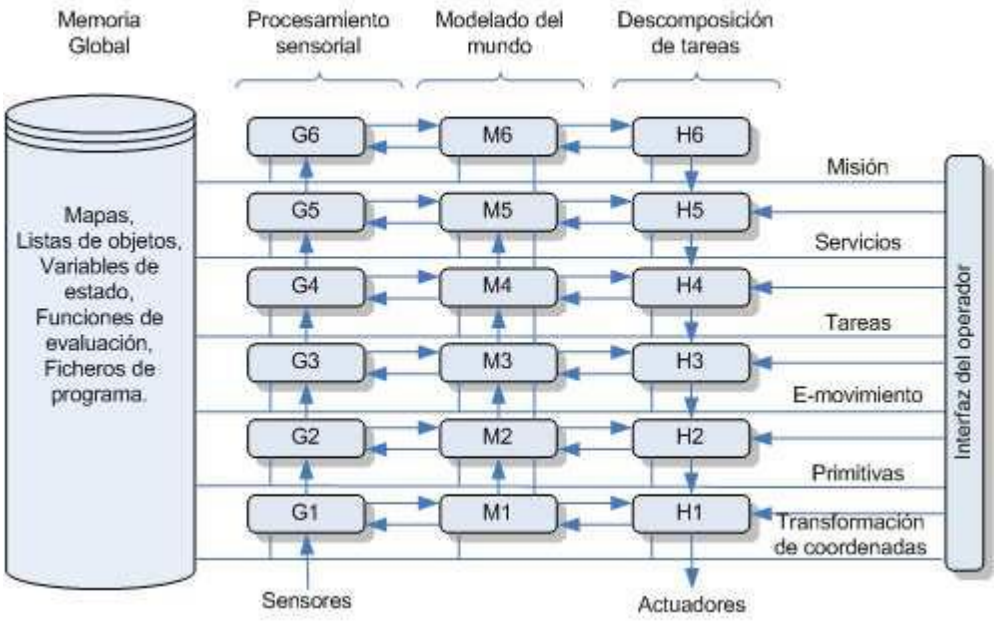


Figura 15: Componentes del modelo paralelo NASREM [18]

Paradigma híbrido

Es el comportamiento de un ser vivo cualquiera, una misma acción puede haberse generado en diversos niveles. El paradigma híbrido, surgido en 1990 [25], incorpora una capa deliberativa sobre la capa reactiva (ver Figura 16). Este paradigma cuenta con la velocidad de reacción y el cumplimiento de las restricciones temporales que aporta el nivel reactivo, y a su vez con la potencia de navegación propia del nivel deliberativo.

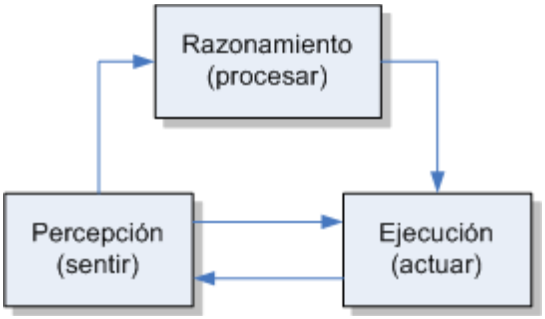


Figura 16: Esquema del paradigma de control híbrido [18]

Por tanto, se considera que el paradigma híbrido está formado por dos capas: una capa reactiva que obtiene información del entorno y que proporciona los mecanismos de supervivencia básica en el entorno al robot, y una capa deliberativa que obtiene información de la capa reactiva y que permite al robot llevar a cabo tareas complejas mediante la interacción en las acciones llevadas a cabo por el robot mediante la conexión con el bloque encargado de ejecutar las acciones situado en la capa reactiva.

Arquitecturas híbridas

Existen gran cantidad de arquitecturas híbridas que pueden ser divididas, por sus características, en organizativas, basadas en jerarquías de estados, orientadas a modelos y organizadas en niveles.

En este apartado se exponen de forma breve algunos ejemplos de cada uno de estos tipos de arquitecturas híbridas.

Arquitecturas híbridas organizativas

Consisten en las arquitecturas que realizan una división de responsabilidades similares a las que se hacen en los negocios. Este tipo de arquitecturas, conocidas también como gerenciales, están formadas por una capa reactiva (con un conocimiento global del mundo) y una deliberativa (con los comportamientos básicos con algo de percepción) claramente diferenciadas. En la capa deliberativa se realiza la organización de responsabilidades mediante el uso de una jerarquía. Un ejemplo de estas arquitecturas es la arquitectura HILARE [26].

- HILARE

La arquitectura HILARE, desarrollada por el LAAS (*Laboratoire d'analyse et d'Architecture des Systèmes*, Toulouse), se compone de tres niveles jerárquicos, que cuentan con unas restricciones temporales y una forma de representar los datos distintos (ver Figura 17).

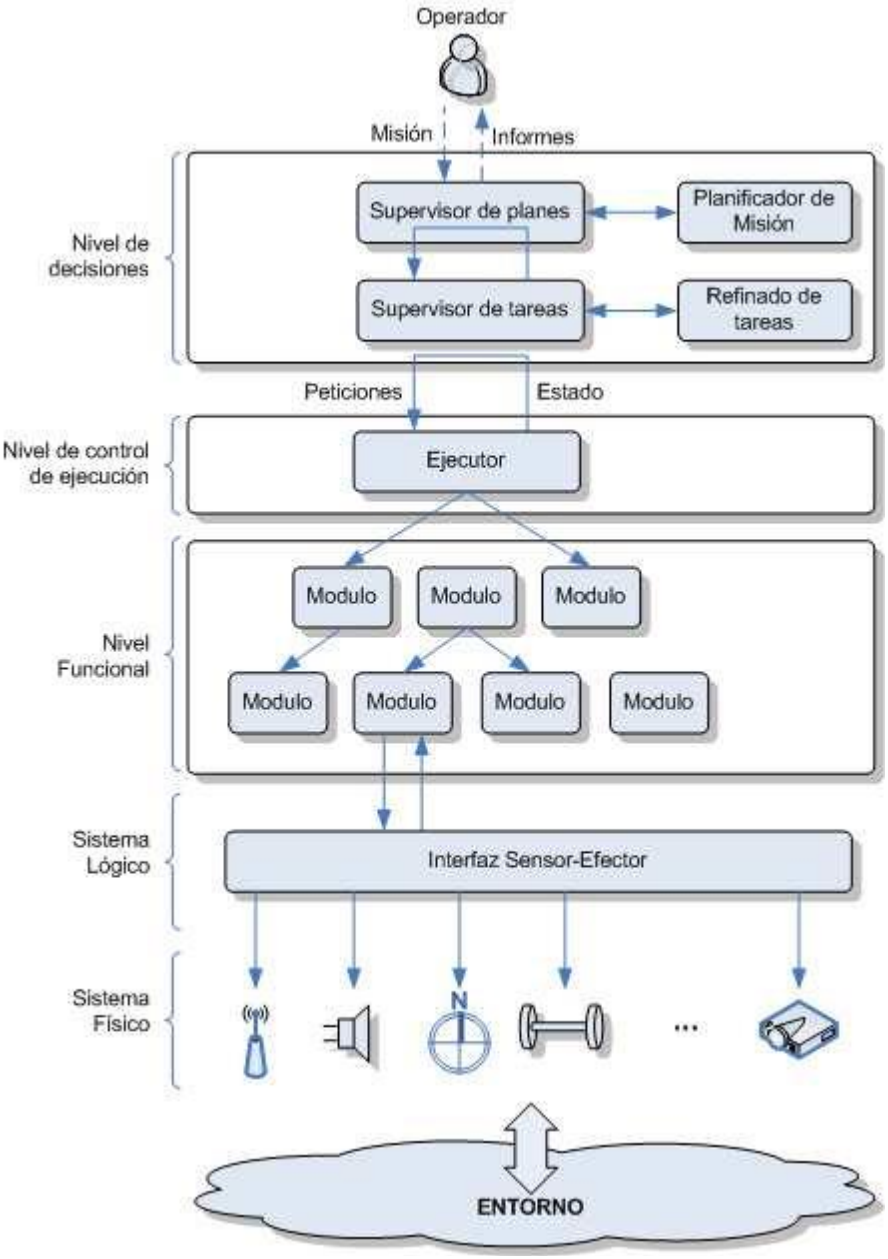


Figura 17: Esquema de la arquitectura HILARE [18]

El nivel más cercano al hardware, y por tanto a la navegación reactiva es el funcional; este nivel es el encargado de controlar tanto la percepción básica del robot como su capacidad básica de reacción. El siguiente nivel es el nivel de ejecución, que se encarga del control y coordinación dinámica a la hora de ejecutar las diferentes funciones que se distribuyen entre los módulos, según las tareas especificadas en el nivel superior. El último nivel es el de decisión, que se encarga de la producción de las tareas del plan a seguir y de la supervisión de la ejecución.

Arquitecturas híbridas basadas en jerarquías de estados

Estas arquitecturas se caracterizan porque las actividades se organizan estrictamente en el ámbito temporal, también llamado "estado de conocimiento". Normalmente son arquitecturas con tres capas, dependientes del estado temporal en el que trabajan, dedicadas al pasado, presente y futuro. Estas arquitecturas distinguen entre las capas reactiva y deliberativa en función del instante temporal en el que trabajan. La capa deliberativa trabaja con el conocimiento pasado y futuro (suposiciones o predicciones). La organización de esta capa se realiza en función de un estado interno temporal. Por otro lado, los comportamientos suceden mediante la generación y monitorización de secuencias de comportamientos, la unión de estos en lo que se conoce como "habilidades" y la subsunción. Un ejemplo de esta arquitectura es la 3T [27].

- 3T (three tiered)

La arquitectura 3T (*Three Tiered*) fue diseñada para su funcionamiento en varios tipos de robots diferentes, siendo independiente del hardware. Hace uso de varios niveles de abstracción que dan la capacidad al robot de realizar varias tareas. Además, esta arquitectura está diseñada para su funcionamiento en entornos dinámicos. Sus componentes pueden verse en la Figura 18.

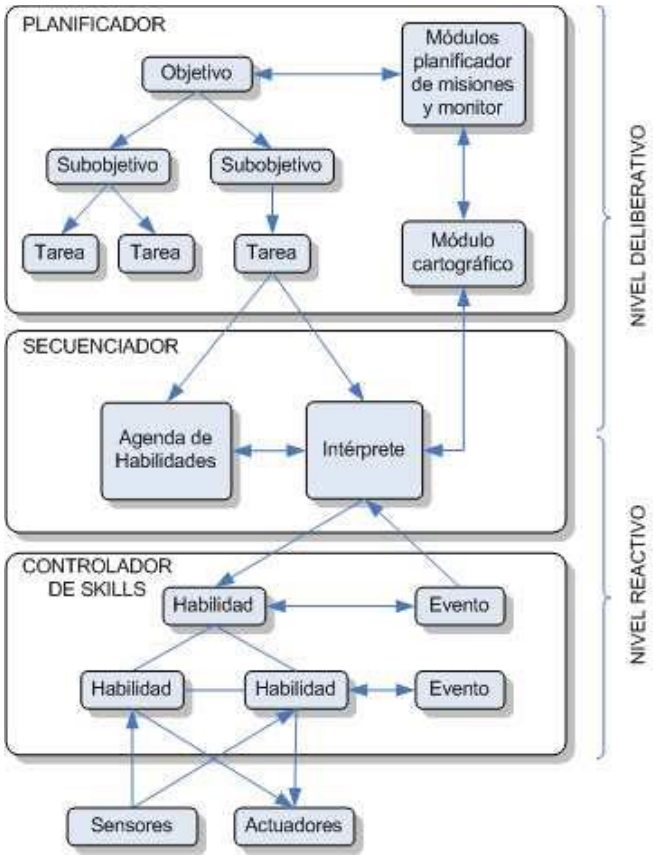


Figura 18: Esquema de la arquitectura de control inteligente 3T [18]

Originalmente, en el nivel superior se encuentra el planificador, que permite seleccionar objetivos. En el nivel intermedio, los objetivos seleccionados son descompuestos en una serie de acciones de bajo nivel. En el nivel inferior, estas acciones son descompuestas de nuevo en "habilidades".

Arquitecturas híbridas orientadas a modelos

Las arquitecturas orientadas a modelos son aquellas en las que, como sensores virtuales, se utilizan modelos. En este tipo de arquitecturas, el nivel deliberativo está compuesto por todo lo que tiene que ver con los comportamientos orientados a la consecución de un objetivo. El nivel reactivo de estas arquitecturas está compuesto por los comportamientos representados como unidades de control que actúan en el presente, pero que pueden hacer uso del conocimiento global a través de sensores virtuales.

La organización del sistema se realiza mediante componentes reactivos y deliberativos, un modelo del mundo y del estado del robot. Dentro de esta arquitectura, los comportamientos surgen mediante la secuencia de comportamientos usando lógica borrosa o por voto. Un ejemplo de esta arquitectura es la arquitectura Saphira [28].

- Saphira

La arquitectura Saphira es un sistema integrado de sensorización y control. Su componente central es el LPS (espacio de percepción local) que es una representación geométrica del entorno del robot. Este componente tiene como función proporcionarle al robot información sobre el entorno en el que se encuentra y es de gran importancia porque sin él no se podrían realizar las tareas de fusión de información, de planificación del movimiento y la integración de la información del mapa.

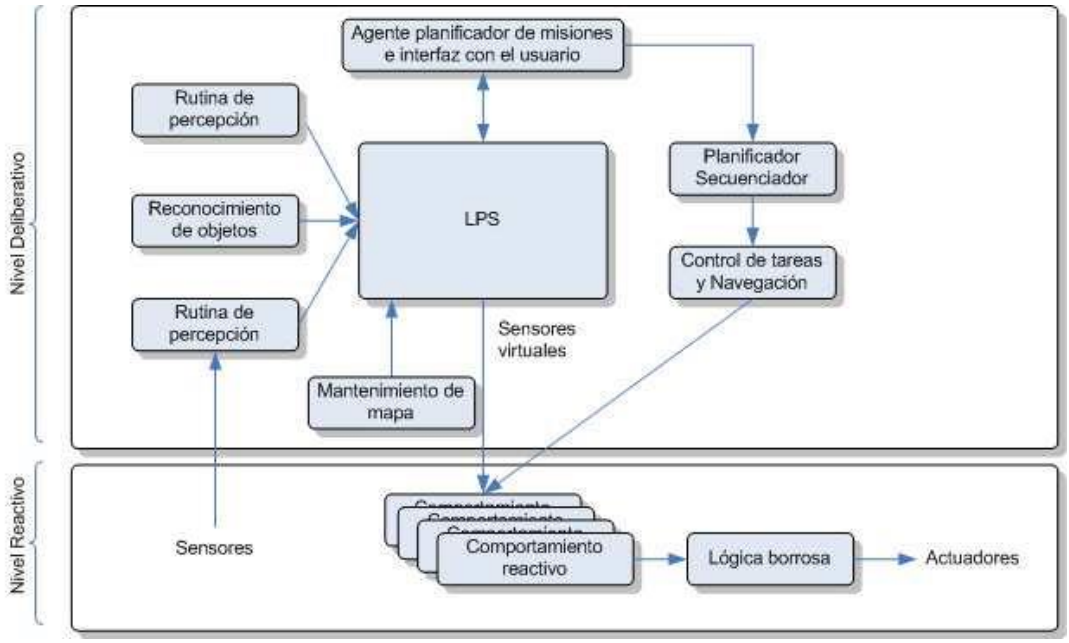


Figura 19: Componentes de la arquitectura de control Saphira [18]

La organización, como se puede observar en la Figura 19, es parcialmente vertical y parcialmente horizontal. El LPS se encarga de procesar la información sensorial que le proporcionan diferentes rutinas de percepción para elaborar la información necesaria para reconocer objetos y navegar por el entorno. El comportamiento reactivo del robot es guiado mediante el uso de comportamientos complejos que realizan acciones dirigidas por objetivos.

Una de las características que diferencian a Saphira es la combinación que se realizan de los comportamientos con técnicas de lógica difusa. Además, los componentes que forman la arquitectura son independientes, lo que implica que su ejecución no tiene por qué llevarse a cabo en el mismo nodo correspondiente al robot sobre el que está actuando.

Arquitecturas híbridas organizadas en niveles

Existen otros modelos de arquitecturas híbridas a parte de los presentados hasta ahora que no pueden ser clasificados bajo ninguno de los anteriores tipos expuestos, pero que tienen en común que se organizan en niveles diferenciados independientes de características espaciales y es interesante mencionarlos. Algunos de estos modelos son la arquitectura GLAIR [29] y [30], la arquitectura Sharp [31], la arquitectura BERRA (*Behaviour-based Robot Research Architecture*) [32], la arquitectura SSS (*Servo, Subsumption, Symbolic*) [33] o la arquitectura de Payton [34].

2.3. Simuladores

Este trabajo tiene como objetivo la implementación de un vehículo autónomo en un entorno simulado, con la intención de que en un futuro se implemente en un entorno real, por tanto, es necesario una introducción a los simuladores, tanto de conducción como de robótica.

"*Un simulador es un aparato que reproduce el comportamiento de un sistema en determinadas condiciones, aplicado generalmente para el entrenamiento de quienes deben manejar dicho sistema*" [35].

El objetivo de un simulador es reproducir tanto las sensaciones físicas como el comportamiento de los elementos del objeto a simular, eliminando el riesgo de accidentes o daños tanto para el equipo como para el usuario. Además, los simuladores permiten poner a prueba los sistemas y comportamientos del objeto simulado y del usuario en todo tipo de entornos.

2.3.1. Simuladores de conducción

Los simuladores de conducción son el tipo de simulador más popular en la actualidad debido a que tiene distintos tipos de usos. Se utilizan simuladores de conducción para la investigación y desarrollo de nuevos sistemas de asistencia a la conducción (los ADAS que se detallan anteriormente en este documento). Estos simuladores son también utilizados en autoescuelas para la formación de conductores, al poder repetir de una forma rápida y sin ningún tipo de riesgo situaciones como adelantamientos, aparcamientos o circulación por rotondas. También son usados para entrenar a los pilotos profesionales de diferentes categorías de deportes de motor como la Formula 1. Además de todo esto, la gran popularidad de la que gozan los simuladores de conducción es debida a su uso en el mercado del entretenimiento

digital, y más concretamente en el mercado de los videojuegos. A continuación, se detallan ejemplos reales de simuladores de cada uno de los ámbitos mencionados.

2.3.1.1. Simuladores de conducción para el desarrollo de sistemas de conducción

A continuación, se detallan algunos ejemplos de simuladores de conducción que son utilizados para investigar y desarrollar sistemas de conducción.

STISIM Drive Simulator

El software del simulador de conducción STISIM Drive [36] es programable y totalmente interactivo. Más de 500 universidades, agencias gubernamentales, instalaciones médicas, centro de entrenamiento y empresas han usado este simulador para llevar a cabo investigaciones, realizar evaluación y tratamientos de terapia ocupacional y realizar entrenamiento de conductores. El software STISIM Drive posee sobre 90 simulaciones de conducción listas para ser ejecutadas que presentan diversas situaciones de conducción, entornos de conducción personalizables y una extensa librería de objetos de carretera.

Figura 20: Simulador de conducción STISIM Drive Simulator [37]

National Advanced Driving Simulator (NADS)

El *National Advanced Driving Simulator* (NADS) [38] es un centro auto sostenido de investigación de seguridad en el transporte en el Parque de Investigación de la Universidad de Iowa, Estados Unidos. NADS hace uso de un conjunto de simuladores de conducción de clase mundial vehículos instrumentados para llevar a cabo estudios de investigación para los sectores público y privado.

El desarrollo y la investigación que se lleva a cabo en el NADS, patrocinado por el gobierno estadounidense mediante la NHTSA, así como por los socios privados, militares y de la industria del automóvil, salva vidas, mejora la calidad de vida de los automovilistas, avanza en el estado del arte de la conducción, y mejora la eficiencia y la productividad de la industria de la automoción.

NADS ofrece simuladores con un rango de fidelidades con el objetivo de lograr de la mejor manera posible los requisitos de cada proyecto: NADS-1, el simulador de mayor fidelidad del mundo; NADS-2, un simulador de base fija; y el simulador NADS MiniSim, un simulador portátil enfocado a computadoras de bajo coste.

Figura 21: Simulador de conducción National Advanced Driving Simulator [38]

2.3.1.2. Simuladores de conducción para la formación de conductores

A continuación, se muestran ejemplos de simuladores de conducción orientados exclusivamente a la formación de conductores.

DRIVESIM

DriveSim [39] es un simulador de conducción para autoescuelas, centro de formación, cursos de prevención de riesgos laborales o cualquier tipo de evento. Este programa de simulación ha sido especialmente diseñado como una herramienta educativa para conductores.

Este simulador cuenta con todo tipo de situaciones y contextos ultra realistas que permiten a los usuarios practicar como si estuviesen al volante de un vehículo real, pero sin los riesgos físicos y materiales que eso supondría. Este simulador incluye conducción con condiciones climatológicas adversas, comportamiento agresivo de los demás conductores o conducción en todo tipo de vías, todo ello representado con gráficos ultra realistas.

Figura 22: Simulador de conducción *DriveSim [39]*

City Car Driving

City Car Driving [40] es un simulador de conducción diseñado para ayudar a los usuarios a experimentar la conducción en una gran ciudad o en un país con diferentes condiciones. Se ha hecho especial énfasis en una variedad de diferentes situaciones en la carretera y una conducción realista. Éste simulador usa avanzadas físicas del automóvil para logar una experiencia realista de conducción y un motor gráfico de alto rendimiento para un alto realismo gráfico. Coches, carreteras, tráfico, peatones, todo esto se ha recreado para que los usuarios sientan que conducen un coche real en una ciudad real. Todos estos elementos son modificables para adaptar la experiencia al gusto del usuario.

Figura 23:Entorno de simulación del simulador de conducción City Car Driving [40]

2.3.1.3. Simuladores de conducción en la industria del entretenimiento

Por último, existen simuladores de conducción que se usan como videojuegos. Pese a tratarse de un simple entretenimiento, el realismo de algunos de estos simuladores es muy alto, llegando a simular con gran exactitud las condiciones que se dan en las carreras profesionales de las competiciones del motor. A continuación, se muestran algunos ejemplos de estos simuladores.

Assetto Corsa

Assetto Corsa [41] es uno de los juegos de carreras más populares del mercado, e incluye los trazados más icónicos del mundo del motor, haciendo uso de un motor gráfico que recrea ambientes inmersivos, iluminación dinámica y superficies y materiales realistas.

Diseñado para proporcionar una experiencia de conducción altamente realista, el avanzado motor de físicas del que hace uso incluye características jamás vistas antes en cualquier otro simulador de carreras como planos en las ruedas, ciclos de calor incluyendo "*graining*" y "*blistering*", una simulación de la aerodinámica muy avanzada, sistemas híbridos y simulación de la recuperación de energía. Este simulador cuenta además con coches licenciados y reproducidos con la mayor precisión gracias a la cooperación oficial con fabricantes de coches.

Figura 24: Captura de pantalla del videojuego de simulación de conducción Asseto Corsa [41]

Project Cars

Project Cars [42] es un videojuego de simulación de conducción que ha sido creado por jugadores, probado por verdaderos conductores de carreras y una de las elecciones preferidas por los profesionales de los deportes electrónicos.

Ofreciendo gráficos de última generación, un innovador tiempo del día dinámico y sistema de clima, y un profundo sistema de modificación del coche y "*pit stop*", los jugadores pueden crear un conductor, seleccionar entren una variedad de deportes de motor, y comenzar a conseguir una serie de objetivos históricos en una misión para entrar en el Salón de la Fama.

Figura 25: Jugador probado el videojuego de simulación de conducción Project Cars [42]

2.3.2. Simuladores de robótica

Aunque el objetivo de este trabajo es realizar una simulación de un vehículo autónomo, dicha simulación no se va a llevar a cabo en un simulador de conducción como los descritos anteriormente, si no que se va a utilizar un simulador de robótica.

El uso de los simuladores de robótica se debe a que la mayoría de robots son caros y si se utilizan para realizar las pruebas durante la fase de desarrollo, pueden causarse daños de cuantiosa valía. En cambio, si el desarrollo se realiza en un simulador, cualquier error en la funcionalidad no supone un coste en el mantenimiento del robot. A continuación, se exponen algunos ejemplos de simuladores de robótica.

2.3.2.1. Microsoft Robotics Developer Studio

Microsoft Robotics Developer Studio (MRDS) [43] es un entorno basado en Windows para uso académico, profesional o aficionado para crear aplicaciones robóticas para variedad de plataformas de hardware. MRDS incluye una librería basada en el lenguaje .NET que consiste en dos componentes: Tiempo de Ejecución Concurrente y Coordinado (CCR) que hace más simple escribir programas para manejar entradas asíncronas de diferentes sensores robóticos y salidas a motores y actuadores, y Servicios de Software Descentralizados (DSS) cuya arquitectura

orientada a servicios hace más simple acceder y responder a los estados de un robot usando un navegador web o una aplicación basada en Windows.

MRDS es también una plataforma escalable y extensible. El modelo de programación de MRDS es aplicable a variedad de robots, permitiendo a los usuarios transferir sus habilidades a través de múltiples plataformas. Se puede conectar de forma remota desde el PC al robot usando un puerto de serie, Bluetooth, Wifi o un modem RF. Los programas desarrollados pueden ser también ejecutados de forma nativa en robot basados en PC ejecutando un Sistema Operativo de la familia Windows, permitiendo un funcionamiento totalmente autónomo.

Las aplicaciones robóticas en este simulador se pueden desarrollar usando el lenguaje C# en Microsoft Visual Studio, pero también cuenta con una herramienta de programación visual (VPL) para que los usuarios que no están familiarizados con la programación puedan también crear sus propias aplicaciones, simplemente arrastrando y soltando bloques que representan servicios, y conectándolos.

Además, MRDS cuenta con una herramienta de simulación gráfica de aplicaciones robóticas en 3D llamada *Microsoft Visual Simulation Environment* (VSE). VSE incluye la tecnología NVIDIA PhysX, que permite la simulación de físicas reales para los modelos robóticos. La última versión del MRDS (MRDS 4) introduce como nueva característica el soporte para el uso del sensor Kinect de Microsoft.

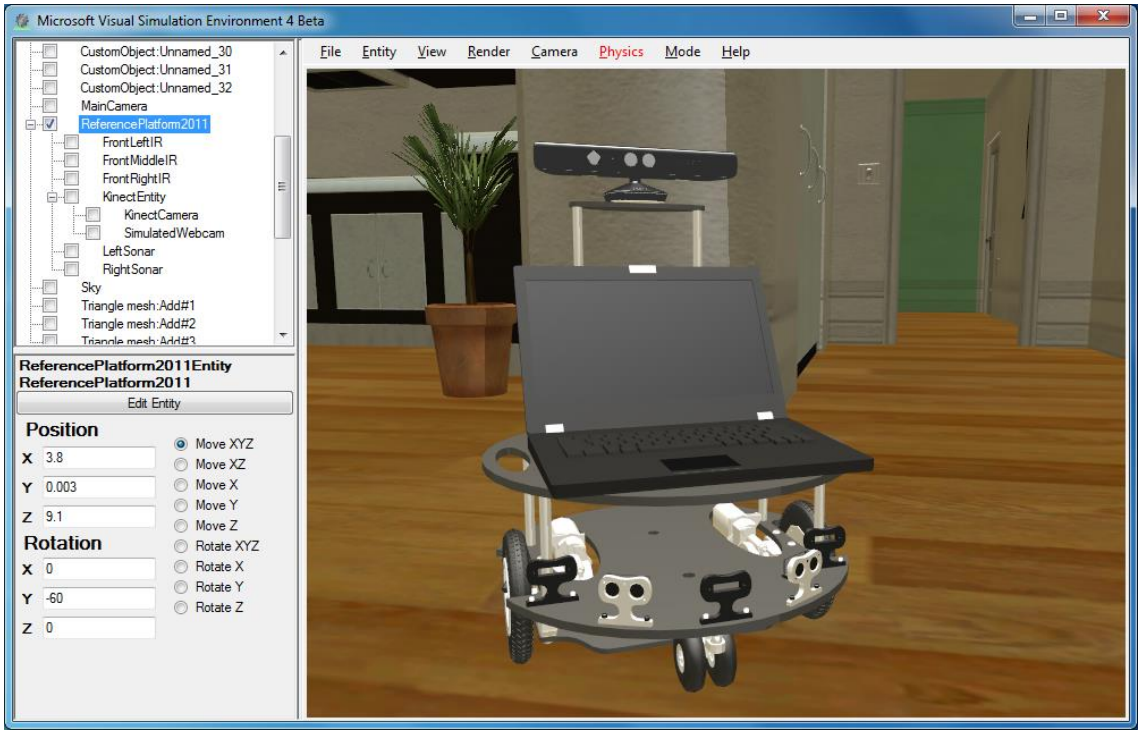


Figura 26: Entorno de simulación Microsoft Robotics Developer Studio [43]

2.3.2.2. RoboWorks

Roboworks [44] es una herramienta software para el modelado 3D, la simulación y animación de cualquier sistema físico, desarrollado por la empresa Newtonium.

La mayoría del software de análisis y visualización como Matlab, MathCad o LabVIEW cuentan con gráficos limitados a tablas y grafos. El software RoboWorks añade a estos paquetes de software la capacidad de modelar y animar en 3D. El usuario puede hacer su análisis en el software que elija y visualizar y animar sus resultados en RoboWorks.

RoboWorks soporta la animación interactiva mediante teclado (el usuario puede animar mientras construye el modelo), a través de un archivo de datos, o mediante RoboTalk. RoboTalk es un software libre que proporciona una interfaz de programación de aplicaciones, ejecutable en cualquier plataforma que soporte TCP/IP para controlar e interactuar con los modelos creados en RoboWorks en tiempo real.

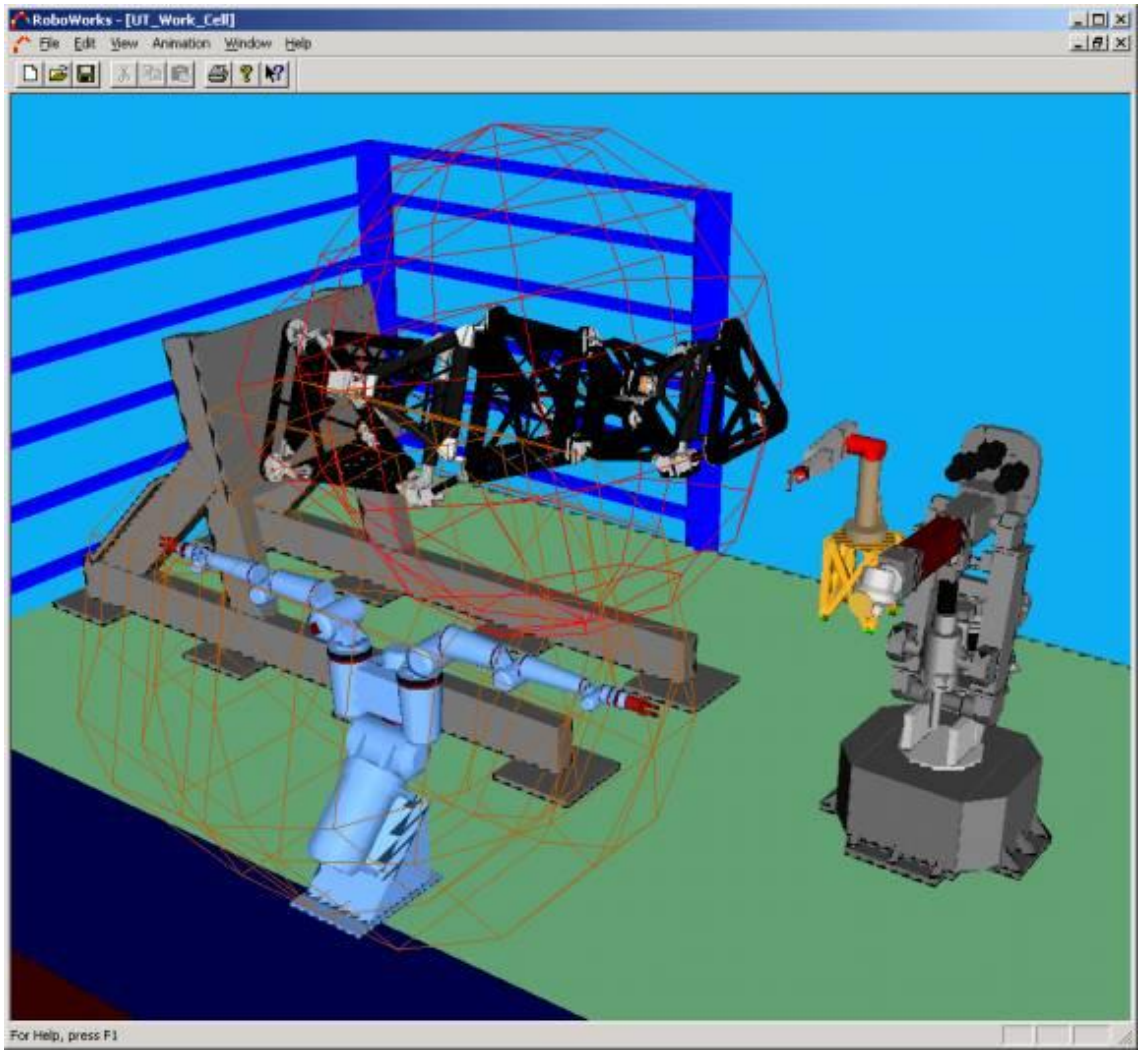


Figura 27:Entorno de simulación Roboworks [44]

2.3.2.3. Webots

Webots [45] es un programa que ofrece herramientas para modelar, programar y simular robots móviles. Webots ofrece al usuario la posibilidad de diseñar complejas simulaciones, uno o más robots de diferentes tipos, en un entorno común. Cada una de las propiedades de los objetos, como su color o su forma, pueden ser elegidas por el usuario. Además, el usuario tiene a su disposición una variada gama de sensores y actuadores simulados con los que puede equipar a sus creaciones. Los controladores de los robots pueden ser programados con el entorno de programación con el que cuenta el propio programa o con entornos de terceras empresas. Una vez diseñados y programados, el comportamiento de los robots puede ser probado en mundos físicamente realistas, para luego trasladar sus controladores a robot reales.

Webots se usa en más de 1245 universidades y centros de estudio de todo el mundo, Al usar Webots, el usuario se beneficia de una tecnología probada que ha sido codesarrollada por el Instituto Federal Suizo de Tecnología de Lausanne, probado a fondo, bien documentado y mantenido de forma continua durante más de 19 años.

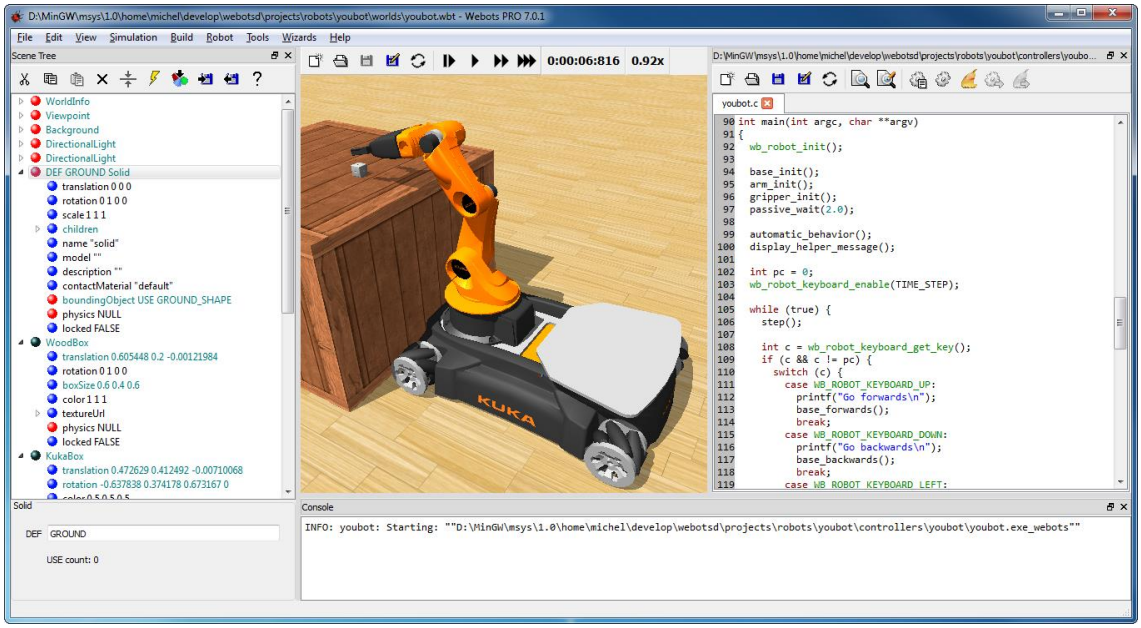


Figura 28: Entorno de simulación Webots [45]

CAPÍTULO 3

Análisis y Diseño del Sistema

3. Análisis y Diseño del Sistema

En esta sección de la memoria del trabajo fin de grado se realiza un análisis detallado del sistema a desarrollar y su diseño. En primer lugar, se detallan las normas y restricciones que debe contemplar el sistema, así como su entorno operacional. A continuación, se especifican los requisitos que debe cumplir el sistema y se detallan sus casos de uso.

3.1. Normas y restricciones del proyecto

Este apartado se divide en dos secciones, las normas que debe cumplir el sistema para poder ser implantado y las restricciones bajo las que debe ser desarrollado.

3.1.1. Especificación de estándares y normas

El objetivo del proyecto es desarrollar un controlador que dirija el funcionamiento de un vehículo autónomo en un entorno simulado, pero que pueda ser usado más adelante en un vehículo físico en un entorno real. Por tanto, se está desarrollando un controlador para un vehículo autónomo que podría circular junto con otros vehículos (de conducción autónoma o de conducción manual). Debido a esto, para poder implantar el controlador en un vehículo físico y circular con él, se debe conseguir la autorización de pruebas o ensayos de investigación realizados con vehículos de conducción automatizada en vías abiertas al tráfico en general, que se encuentra en la Instrucción 15/V-113 de la Dirección General de Tráfico de España [46] [47].

3.1.2. Restricciones generales

Las restricciones que debe cumplir el sistema son las siguientes:

- El trabajo con la simulación se debe realizar con el programa Webots en su versión 7.
- El código del controlador debe desarrollarse en el lenguaje de programación C++.
- Para la ejecución del controlador mediante ROS, se debe contar con un sistema operativo basado en Linux.
- El ordenador con sistema operativo Linux debe tener instalado ROS, en su versión Jade, para poder ejecutar el controlador.
- El código específico de ROS del controlador debe estar desarrollado en el lenguaje de programación C++.

3.2. Entorno operacional

El sistema con el que se ha llevado a cabo el desarrollo de del proyecto tienen la siguiente configuración software y hardware:

- Ordenador con procesador Intel Core i7-4790 @ 3.60GHz y 16 GB de memoria RAM.
- Sistema Operativo Windows 10 de 64 bits
- Sistema Operativo Ubuntu 14.04 LTS

- Webots versión 7.4.3.
- SketchUp Pro 2016
- Paquete ofimático de Microsoft Office 2016
- ROS versión Jade
- Gantt Project versión 2.8.1.
- StarUML versión 2.7.0.

3.3. Especificación de requisitos

En esta sección de la memoria se detallan los diferentes tipos de requisitos asociados a este proyecto. Cada uno de estos requisitos se va a describir mediante una tabla con el siguiente formato:

RY-X			
PRIORIDAD	☐ Alta	☐ Media	☐ Baja
NECESIDAD	☐ Esencial	☐ Deseable	☐ Opcional
ESTABILIDAD	☐ Alta	☐ Media	☐ Baja
DESCRIPCIÓN			

Tabla 2: Formato de las tablas de requisitos

Dónde:

- **RY-X:** identificador del requisito, siendo:
  - **R:** requisito.
  - **Y:** requisito funcional ('F') o requisito no funcional ('NF').
  - **X:** número entero único que identifica al requisito.
- **PRIORIDAD:** indica la prioridad de cumplimiento del requisito, pudiendo ser ésta de tres niveles: prioridad alta, prioridad media o prioridad baja. Este dato es útil para la planificación del proyecto.
- **NECESIDAD:** indica la importancia del requisito para el desarrollador. Las tres posibles opciones para la necesidad son:
  - **Esencial:** su cumplimiento es imprescindible para el desarrollo del proyecto. Si estos requisitos no se cumplen, el proyecto no puede ser llevado a cabo.
  - **Deseable:** su cumplimiento es esperado, pero no puede asegurarse por algún motivo.
  - **Opcional:** su cumplimiento no es estrictamente necesario, por lo que el hecho de que se cumpla o no se cumpla no afecta en gran medida al desarrollo del proyecto en su totalidad.
- **ESTABILIDAD:** indica lo susceptible que es un requisito a modificaciones durante el desarrollo del proyecto. Cuanto más alta es la estabilidad, menor probabilidad hay de que el requisito sea modificado.

3.3.1. Requisitos funcionales

A continuación, se detallan los requisitos funcionales de este proyecto, que describen los comportamientos que debe tener el sistema a desarrollar.

RF-1			
PRIORIDAD	☒ Alta	☐ Media	☐ Baja
NECESIDAD	☒ Esencial	☐ Deseable	☐ Opcional
ESTABILIDAD	☒ Alta	☐ Media	☐ Baja
DESCRIPCIÓN	El sistema llevará el vehículo autónomo desde el punto en el que se encuentra hasta un punto dado por el usuario.		

Tabla 3: Requisito funcional RF-1

RF-2			
PRIORIDAD	☒ Alta	☐ Media	☐ Baja
NECESIDAD	☒ Esencial	☐ Deseable	☐ Opcional
ESTABILIDAD	☒ Alta	☐ Media	☐ Baja
DESCRIPCIÓN	El sistema permitirá al usuario establecer los puntos de ruta por los que puede circular el vehículo dentro de la simulación.		

Tabla 4: Requisito funcional RF-2

RF-3			
PRIORIDAD	☒ Alta	☐ Media	☐ Baja
NECESIDAD	☒ Esencial	☐ Deseable	☐ Opcional
ESTABILIDAD	☒ Alta	☐ Media	☐ Baja
DESCRIPCIÓN	El sistema permitirá al usuario establecer los puntos de destino a los que puede ir el vehículo dentro de la simulación.		

Tabla 5: Requisito funcional RF-3

RF-4			
PRIORIDAD	☒ Alta	☐ Media	☐ Baja
NECESIDAD	☒ Esencial	☐ Deseable	☐ Opcional
ESTABILIDAD	☒ Alta	☐ Media	☐ Baja
DESCRIPCIÓN	El sistema usará unos puntos de ruta definidos previamente para realizar el recorrido del vehículo.		

Tabla 6: Requisito funcional RF-4

RF-5			
PRIORIDAD	☒ Alta	☐ Media	☐ Baja
NECESIDAD	☒ Esencial	☐ Deseable	☐ Opcional
ESTABILIDAD	☐ Alta	☒ Media	☐ Baja
DESCRIPCIÓN	El sistema buscará el camino entre la posición actual del vehículo y el objetivo mediante un algoritmo de búsqueda.		

Tabla 7: Requisito funcional RF-5

RF-6			
PRIORIDAD	☒ Alta	☐ Media	☐ Baja
NECESIDAD	☐ Esencial	☒ Deseable	☐ Opcional
ESTABILIDAD	☒ Alta	☐ Media	☐ Baja
DESCRIPCIÓN	El sistema detendrá el vehículo cuando éste haya alcanzado el objetivo seleccionado.		

Tabla 8: Requisito funcional RF-6

RF-7			
PRIORIDAD	☒ Alta	☐ Media	☐ Baja
NECESIDAD	☒ Esencial	☐ Deseable	☐ Opcional
ESTABILIDAD	☒ Alta	☐ Media	☐ Baja
DESCRIPCIÓN	El sistema calculará el ángulo del vehículo respecto a los obstáculos que encuentre en el camino.		

Tabla 9: Requisito funcional RF-7

RF-8			
PRIORIDAD	☒ Alta	☐ Media	☐ Baja
NECESIDAD	☒ Esencial	☐ Deseable	☐ Opcional
ESTABILIDAD	☒ Alta	☐ Media	☐ Baja
DESCRIPCIÓN	El sistema modificará el ángulo de las ruedas del vehículo para evitar impactar con los obstáculos detectados.		

Tabla 10: Requisito funcional RF-8

RF-9			
PRIORIDAD	☒ Alta	☐ Media	☐ Baja
NECESIDAD	☒ Esencial	☐ Deseable	☐ Opcional
ESTABILIDAD	☐ Alta	☒ Media	☐ Baja
DESCRIPCIÓN	El sistema admitirá como posición objetivo las coordenadas de los edificios que forman el campus simulado.		

Tabla 11: Requisito funcional RF-9

RF-10			
PRIORIDAD	☐ Alta	☒ Media	☐ Baja
NECESIDAD	☐ Esencial	☒ Deseable	☐ Opcional
ESTABILIDAD	☐ Alta	☒ Media	☐ Baja
DESCRIPCIÓN	El sistema admitirá una única posición como objetivo.		

Tabla 12: Requisito funcional RF-10

RF-11			
PRIORIDAD	☐ Alta	☒ Media	☐ Baja
NECESIDAD	☐ Esencial	☒ Deseable	☐ Opcional
ESTABILIDAD	☐ Alta	☒ Media	☐ Baja
DESCRIPCIÓN	El sistema permitirá al usuario establecer la velocidad a la que debe circular el vehículo.		

Tabla 13: Requisito funcional RF-11

RF-12			
PRIORIDAD	☐ Alta	☒ Media	☐ Baja
NECESIDAD	☐ Esencial	☒ Deseable	☐ Opcional
ESTABILIDAD	☐ Alta	☒ Media	☐ Baja
DESCRIPCIÓN	El sistema evitará que el vehículo sobrepase la velocidad máxima de circulación.		

Tabla 14: Requisito funcional RF-12

RF-13			
PRIORIDAD	☐ Alta	☒ Media	☐ Baja
NECESIDAD	☐ Esencial	☒ Deseable	☐ Opcional
ESTABILIDAD	☐ Alta	☒ Media	☐ Baja
DESCRIPCIÓN	El sistema ejecutará la funcionalidad principal en un intervalo de tiempo determinado.		

Tabla 15: Requisito funcional RF-13

RF-14			
PRIORIDAD	☒ Alta	☐ Media	☐ Baja
NECESIDAD	☒ Esencial	☐ Deseable	☐ Opcional
ESTABILIDAD	☒ Alta	☐ Media	☐ Baja
DESCRIPCIÓN	El sistema creará un nodo (en ROS) para el controlador que se conectará al Nodo Maestro de ROS.		

Tabla 16: Requisito funcional RF-14

RF-15			
PRIORIDAD	☒ Alta	☐ Media	☐ Baja
NECESIDAD	☒ Esencial	☐ Deseable	☐ Opcional
ESTABILIDAD	☒ Alta	☐ Media	☐ Baja
DESCRIPCIÓN	El sistema creará un nodo (en ROS) para el vehículo autónomo que se conectará al Nodo Maestro de ROS.		

Tabla 17: Requisito funcional RF-15

RF-16			
PRIORIDAD	☒ Alta	☐ Media	☐ Baja
NECESIDAD	☒ Esencial	☐ Deseable	☐ Opcional
ESTABILIDAD	☒ Alta	☐ Media	☐ Baja
DESCRIPCIÓN	El sistema creará hilos (en ROS) para cada uno de los dispositivos del vehículo que deben publicar la información que recogen.		

Tabla 18: Requisito funcional RF-16

RF-17			
PRIORIDAD	☒ Alta	☐ Media	☐ Baja
NECESIDAD	☒ Esencial	☐ Deseable	☐ Opcional
ESTABILIDAD	☒ Alta	☐ Media	☐ Baja
DESCRIPCIÓN	El sistema comunicará los nodos (en ROS) mediante los hilos de los dispositivos y peticiones de servicio.		

Tabla 19: Requisito funcional RF-17

3.3.2. Requisitos no funcionales

A continuación, se detallan los requisitos no funcionales de este proyecto, que especifican las características del funcionamiento del sistema.

RNF-1			
PRIORIDAD	☒ Alta	☐ Media	☐ Baja
NECESIDAD	☒ Esencial	☐ Deseable	☐ Opcional
ESTABILIDAD	☒ Alta	☐ Media	☐ Baja
DESCRIPCIÓN	El entorno en el que se ejecute el controlador del vehículo autónomo debe ser una simulación del campus de la Escuela Politécnica de la Universidad Carlos III de Madrid.		

Tabla 20: Requisito no funcional RNF-1

RNF-2			
PRIORIDAD	☐ Alta	☒ Media	☐ Baja
NECESIDAD	☐ Esencial	☒ Deseable	☐ Opcional
ESTABILIDAD	☐ Alta	☒ Media	☐ Baja
DESCRIPCIÓN	El vehículo autónomo utilizado debe ser una simulación de un carro de golf EZ-GO de la serie TXT.		

Tabla 21: Requisito no funcional RNF-2

RNF-3			
PRIORIDAD	☒ Alta	☐ Media	☐ Baja
NECESIDAD	☒ Esencial	☐ Deseable	☐ Opcional
ESTABILIDAD	☒ Alta	☐ Media	☐ Baja
DESCRIPCIÓN	El archivo que define la simulación completa debe tener la extensión .wbt (extensión de los archivos que definen un mundo en Webots).		

Tabla 22: Requisito no funcional RNF-3

RNF-4			
PRIORIDAD	☒ Alta	☐ Media	☐ Baja
NECESIDAD	☒ Esencial	☐ Deseable	☐ Opcional
ESTABILIDAD	☒ Alta	☐ Media	☐ Baja
DESCRIPCIÓN	El archivo que define el vehículo simulado debe tener la extensión .wbo (extensión de los archivos que definen un objeto en Webots).		

Tabla 23: Requisito no funcional RNF-4

RNF-5			
PRIORIDAD	☒ Alta	☐ Media	☐ Baja
NECESIDAD	☒ Esencial	☐ Deseable	☐ Opcional
ESTABILIDAD	☒ Alta	☐ Media	☐ Baja
DESCRIPCIÓN	El archivo que define el suelo del campus simulado debe tener la extensión .wbo (extensión de los archivos que definen un objeto en Webots).		

Tabla 24: Requisito no funcional RNF-5

RNF-6			
PRIORIDAD	☒ Alta	☐ Media	☐ Baja
NECESIDAD	☒ Esencial	☐ Deseable	☐ Opcional
ESTABILIDAD	☒ Alta	☐ Media	☐ Baja
DESCRIPCIÓN	El programador debe establecer los puntos de ruta por los que debe circular el vehículo.		

Tabla 25: Requisito no funcional RNF-6

RNF-7			
PRIORIDAD	☒ Alta	☐ Media	☐ Baja
NECESIDAD	☒ Esencial	☐ Deseable	☐ Opcional
ESTABILIDAD	☒ Alta	☐ Media	☐ Baja
DESCRIPCIÓN	Los puntos de ruta se establecerán mediante sus coordenadas X y Z dentro de la simulación.		

Tabla 26: Requisito no funcional RNF-7

RNF-8			
PRIORIDAD	☒ Alta	☐ Media	☐ Baja
NECESIDAD	☐ Esencial	☒ Deseable	☐ Opcional
ESTABILIDAD	☐ Alta	☒ Media	☐ Baja
DESCRIPCIÓN	El programador distinguirá entre los puntos de ruta usados para la circulación del vehículo y los que identifican a los edificios que son usados como puntos de destino.		

Tabla 27: Requisito no funcional RNF-8

RNF-9			
PRIORIDAD	☐ Alta	☒ Media	☐ Baja
NECESIDAD	☐ Esencial	☒ Deseable	☐ Opcional
ESTABILIDAD	☐ Alta	☒ Media	☐ Baja
DESCRIPCIÓN	El programador establecerá cual es la velocidad máxima para la circulación del vehículo autónomo en la simulación.		

Tabla 28: Requisito no funcional RNF-9

RNF-10			
PRIORIDAD	☐ Alta	☒ Media	☐ Baja
NECESIDAD	☐ Esencial	☒ Deseable	☐ Opcional
ESTABILIDAD	☐ Alta	☒ Media	☐ Baja
DESCRIPCIÓN	El programador establecerá el intervalo de tiempo entre ejecuciones del programa principal.		

Tabla 29: Requisito no funcional RNF-10

RNF-11			
PRIORIDAD	☒ Alta	☐ Media	☐ Baja
NECESIDAD	☒ Esencial	☐ Deseable	☐ Opcional
ESTABILIDAD	☒ Alta	☐ Media	☐ Baja
DESCRIPCIÓN	El lenguaje de programación utilizado para desarrollar el proyecto será C++.		

Tabla 30: Requisito no funcional RNF-11

RNF-12			
PRIORIDAD	☐ Alta	☒ Media	☐ Baja
NECESIDAD	☐ Esencial	☒ Deseable	☐ Opcional
ESTABILIDAD	☒ Alta	☐ Media	☐ Baja
DESCRIPCIÓN	El programador deberá generar un manual de usuario donde se explique de forma clara como hacer funcionar la simulación.		

Tabla 31: Requisito no funcional RNF-12

RNF-13			
PRIORIDAD	☒ Alta	☐ Media	☐ Baja
NECESIDAD	☒ Esencial	☐ Deseable	☐ Opcional
ESTABILIDAD	☒ Alta	☐ Media	☐ Baja
DESCRIPCIÓN	El sistema debe funcionar en la versión 7 (siete) del programa Webots.		

Tabla 32: Requisito no funcional RNF-13

RNF-14			
PRIORIDAD	☒ Alta	☐ Media	☐ Baja
NECESIDAD	☒ Esencial	☐ Deseable	☐ Opcional
ESTABILIDAD	☒ Alta	☐ Media	☐ Baja
DESCRIPCIÓN	El sistema debe funcionar con la versión Jade de ROS.		

Tabla 33: Requisito no funcional RNF-14

3.4. Casos de uso

En esta sección de la memoria se detallan los casos de uso del sistema. Un caso de uso es una descripción de los pasos que debe seguir un actor (quien realiza la acción) para llevar a cabo una tarea concreta del sistema. Los casos de uso pueden describirse de dos formas, de forma tabular o de forma gráfica. A continuación, se detallan los casos de uso de ambas formas para facilitar la comprensión y análisis de estos.

3.4.1. Descripción tabular de casos de uso

Para describir los casos de uso de forma tabular, se usan tablas con el siguiente formato:

CU-X: NOMBRE	
ACTOR	
OBJETIVO	
PRECONDICIONES	
POSTCONDICIONES	
ESCENARIO DE ÉXITO	

Tabla 34: Formato de las tablas de casos de uso

Donde:

- **CU-X: NOMBRE:** identificador del caso de uso, siendo:
  - **CU:** caso de uso.
  - **X:** número entero único que identifica al caso de uso.
  - **NOMBRE:** nombre con el que se identifica al caso de uso.

- **ACTOR:** tipo de usuario que lleva a cabo las acciones descritas en el caso de uso.
- **OBJETIVO:** descripción de la finalidad del caso de uso.
- **PRECONDICIONES:** condiciones que deben cumplirse previas a la realización del caso de uso descrito.
- **POSTCONDICIONES:** condiciones que deben cumplirse una vez llevadas a cabo las acciones que describen el caso de uso.
- **ESCENARIO DE ÉXITO:** pasos que deben seguirse para realizar de forma exitosa el caso de uso.

A continuación, se detallan los casos de uso en forma tabular de este proyecto:

CU-1: Establecer puntos de ruta	
ACTOR	Programador
OBJETIVO	Establecer los puntos de ruta por los que puede pasar el vehículo.
PRECONDICIONES	• El entorno donde se va a desarrollar la simulación debe estar completo. • Deben poderse obtener coordenadas de puntos dentro de la simulación.
POSTCONDICIONES	• Los puntos de ruta quedan definidos mediante sus coordenadas y un nombre específico.
ESCENARIO DE ÉXITO	• Se determinan que posiciones van a ser puntos de ruta dentro de la simulación. • Se comprueban las coordenadas de los puntos de ruta elegidos. • Se modifica el código del controlador con las coordenadas de los puntos de ruta y su identificador.

Tabla 35: Caso de uso CU-1

CU-2: Establecer puntos de destino	
ACTOR	Programador
OBJETIVO	Establecer los posibles puntos de destino que puede tener el vehículo autónomo.
PRECONDICIONES	• El entorno donde se va a desarrollar la simulación debe estar completo. • Los edificios que forman la simulación deben estar correctamente situados en el entorno. • Las coordenadas donde se sitúan los edificios deben ser accesibles.
POSTCONDICIONES	• Los puntos de destino quedan definidos mediante sus coordenadas y un nombre especifico.
ESCENARIO DE ÉXITO	• Se determinan que edificios van a ser puntos de destino dentro de la simulación. • Se comprueban las coordenadas de los puntos de destino elegidos. • Se modifica el código del controlador con las coordenadas de los puntos de destino y su identificador.

Tabla 36: Caso de uso CU-2

CU-3: Establecer velocidad del vehículo	
ACTOR	Programador
OBJETIVO	Establecer la velocidad a la que se desplazará el vehículo durante la simulación.
PRECONDICIONES	• El entorno donde se va a desarrollar la simulación debe estar completo. • El vehículo autónomo simulado debe ser totalmente operativo.
POSTCONDICIONES	• La velocidad a la que se desplazará el vehículo autónomo queda establecida.
ESCENARIO DE ÉXITO	• Se elige la velocidad a la que se quiere que circule el vehículo autónomo durante la simulación. • Se modifica el código del controlador con la velocidad elegida.

Tabla 37: Caso de uso CU-3

CU-4: Ejecutar la simulación	
ACTOR	Programador
OBJETIVO	Ejecutar la simulación en la que el vehículo autónomo se dirige desde su posición hasta una posición objetivo determinada.
PRECONDICIONES	• El entorno donde se va a desarrollar la simulación debe estar completo. • El vehículo autónomo debe ser totalmente operativo. • El vehículo autónomo debe encontrarse en alguna zona del entorno simulado. • Debe haberse seleccionado una posición de destino dentro del entorno simulado. • Debe haberse establecido una velocidad para el vehículo autónomo. • Deben haberse establecido los puntos de ruta por los que debe pasar el vehículo.
POSTCONDICIONES	• El vehículo autónomo se desplaza desde su posición inicial hasta la posición objetivo sin colisionar con ningún obstáculo.
ESCENARIO DE ÉXITO	• Se determina la posición de destino de la simulación. • Se establece la velocidad a la que circulará el vehículo autónomo. • Se ejecuta la simulación. • La simulación se detiene cuando el vehículo autónomo alcanza su objetivo.

Tabla 38: Caso de uso CU-4

CU-5: Iniciar ROS	
ACTOR	Programador
OBJETIVO	Iniciar el Nodo Maestro de ROS para poder ejecutar el controlador del vehículo autónomo.
PRECONDICIONES	• Debe estar instala la versión Jade de ROS. • El entorno de ROS debe haberse configurado correctamente para su uso.
POSTCONDICIONES	• El nodo maestro de ROS se conecta.
ESCENARIO DE ÉXITO	• Se ejecuta la orden para la conexión del Nodo Maestro de ROS. • El Nodo Maestro de ROS se conecta correctamente.

Tabla 39: Caso de uso CU-5

CU-6: Ejecutar nodo controlador de ROS	
ACTOR	Programador
OBJETIVO	Ejecutar el nodo de ROS correspondiente al controlador del vehículo autónomo.
PRECONDICIONES	• El nodo maestro de ROS ha sido conectado. • La simulación se está ejecutando en Webots. • El entorno donde se va a desarrollar la simulación debe estar completo. • El vehículo autónomo debe ser totalmente operativo. • El vehículo autónomo debe encontrarse en alguna zona del entorno simulado. • Debe haberse seleccionado una posición de destino dentro del entorno simulado. • Debe haberse establecido una velocidad para el vehículo autónomo. • Deben haberse establecido los puntos de ruta por los que debe pasar el vehículo.
POSTCONDICIONES	• El vehículo autónomo se desplaza desde su posición inicial hasta la posición objetivo sin colisionar con ningún obstáculo.
ESCENARIO DE ÉXITO	• Se inicia el nodo maestro de ROS. • Se ejecuta la simulación. • Se determina la posición de destino de la simulación. • Se establece la velocidad a la que circulará el vehículo autónomo. • Se ejecuta el nodo del controlador. • El vehículo se detiene cuando el vehículo autónomo alcanza su objetivo.

Tabla 40: Caso de uso CU-6

3.4.2. Descripción gráfica de casos de uso

A continuación, se detallan de forma gráfica los casos de uso descritos en el apartado anterior:

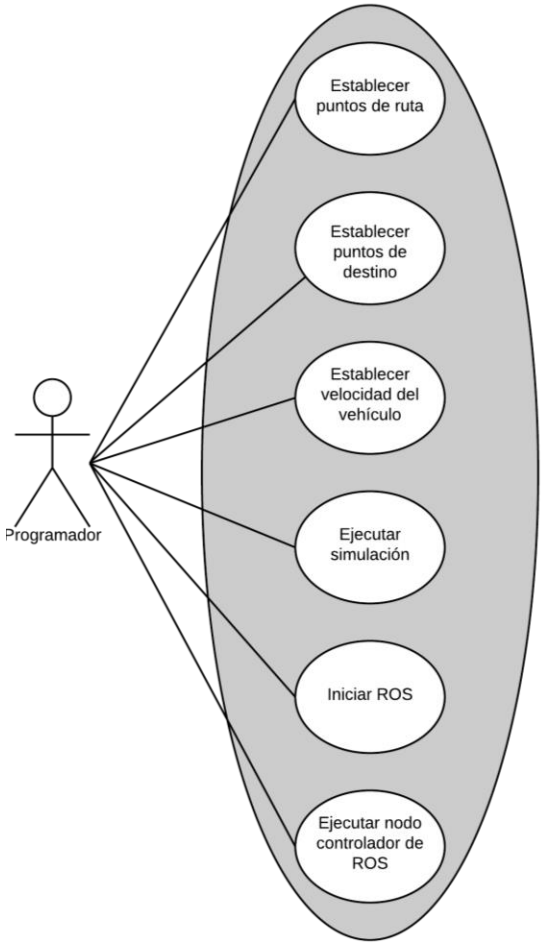


Figura 29: Descripción gráfica de los casos de uso

3.5. Matriz de trazabilidad

En la Tabla 41 se muestra la matriz de trazabilidad entre requisitos funcionales y casos de uso. En esta matriz se indican los casos de uso que cubren cada uno de los requisitos funcionales propuestos.

	CASOS DE USO					
REQUISITOS	**CU-1**	**CU-2**	**CU-3**	**CU-4**	**CU-5**	**CU-6**
RF-1				X		X
RF-2	X					
RF-3		X				
RF-4				X		X
RF-5				X		X
RF-6				X		X
RF-7				X		X
RF-8				X		X
RF-9				X		X
RF-10				X		X
RF-11			X			
RF-12				X		X
RF-13				X		X
RF-14					X	
RF-15					X	
RF-16						X
RF-17						X

Tabla 41: Matriz de trazabilidad

CAPÍTULO 4

Elementos de la simulación

4. Elementos de la simulación

En esta sección de la memoria se describen los elementos gráficos que forman parte de la simulación en la que desarrolla este trabajo, así como las herramientas utilizadas para llevar a cabo el diseño de dichos elementos. Los elementos principales que forman la simulación son el campus de la Escuela Politécnica Superior de la Universidad Carlos III de Madrid y el vehículo autónomo que circula por dicho campus simulado.

4.1. SketchUp

La herramienta utilizada para llevar a cabo el modelado de los elementos del campus de la Escuela Politécnica de la Universidad Carlos III de Madrid de forma detallada ha sido SketchUp.

SketchUp [48] es un software de diseño gráfico y modelado en tres dimensiones basado en caras que permite modelar de forma sencilla cualquier tipo de objeto. Se ha elegido este software porque se trata de una herramienta de modelado 3D muy potente, pero a su vez muy intuitiva y fácil de manejar. Además, existe la posibilidad de exportar todos los diseños que se hagan en SketchUp como archivos de tipo VRML (*Virtual Reality Modeling Language*) [49], pudiéndolos importar después directamente desde el programa Webots (que utiliza también este tipo de archivos para definir los elementos de sus simulaciones), donde se lleva a cabo la simulación. Esta compatibilidad entre programas facilita en gran manera el trabajo de modelado y simulación de este proyecto.

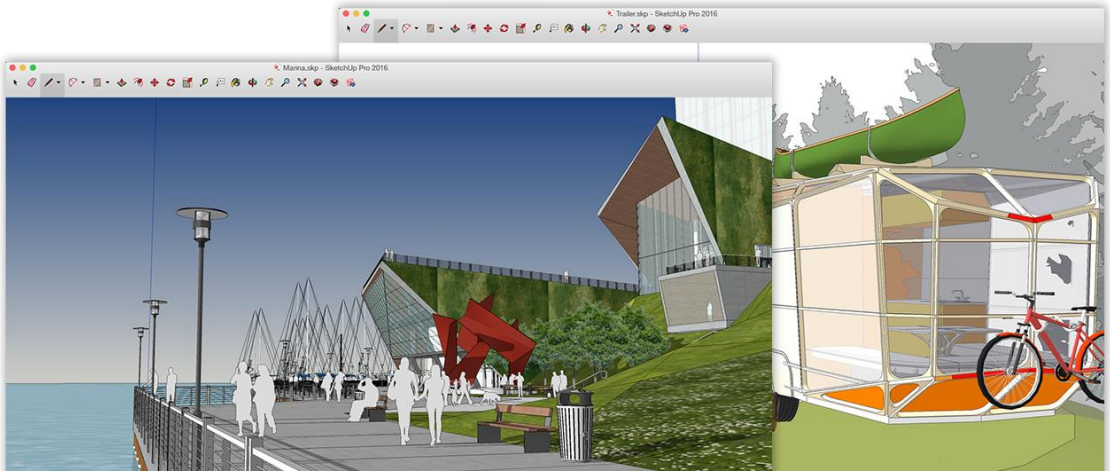


Figura 30: Software de diseño gráfico SketchUp [48]

4.1.1. Campus EPS Carlos III

El entorno en el que se llevará a cabo la simulación y por el que circulará el vehículo autónomo será una representación del campus de Escuela Politécnica Superior de la Universidad Carlos III de Madrid, situado en la población de Leganés. Este entorno simulado se puede dividir en dos partes: los edificios que forman el campus y el suelo de dicho campus.

Figura 31: Plano del campus de la Escuela Politécnica de la Universidad Carlos III de Madrid [50]

4.1.1.1. Edificios del campus

Los edificios simulados del campus son los siguientes (los números que acompañan a los edificios se corresponden con los números asignados a éstos en la Figura 31):

1. Edificio San Agustín de Betancourt
2. Edificio Sabatini
3. Edificio Rey Pastor
4. Edificio Torres Quevedo
5. Edificio Padre Soler
6. Centro Deportivo Alfredo Di Stefano
7. Edificio Juan Benet

8. Torre de la caldera

La simulación de estos edificios del campus fue realizada por Don Víctor Romero Pérez para su trabajo fin de grado "Creación de un entorno 3d para la simulación de tráfico urbano" [51] y reutilizados para este proyecto.

A continuación, se muestran algunos ejemplos de las simulaciones de estos edificios. En el "Anexo B. Modelos del campus" de este documento se muestra de forma más extensa las simulaciones de los edificios del campus.

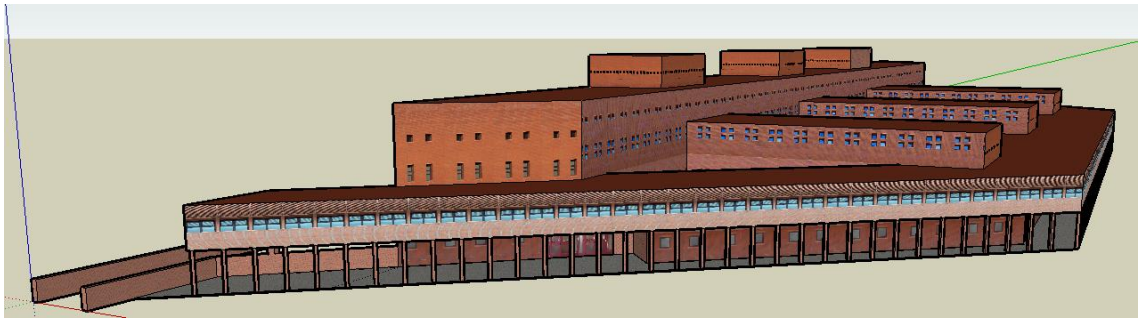


Figura 32: Simulación del edificio San Agustín de Betancourt

Figura 33: Simulación del Edificio Torres Quevedo

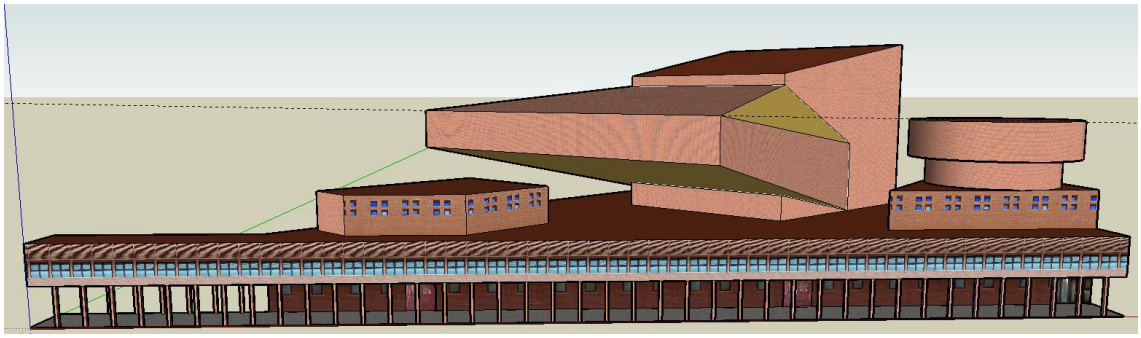


Figura 34: Simulación del edificio Padre Soler

4.1.1.2. Suelo del campus

La superficie del campus ha sido elaborada partiendo de cero mediante el programa de modelado SketchUp. La imagen del suelo del campus de la que se ha partido es la siguiente:

Figura 35: Vista cenital por satélite del campus de la EPS

El resultado del modelado del suelo del campus puede observarse en las siguientes imágenes:

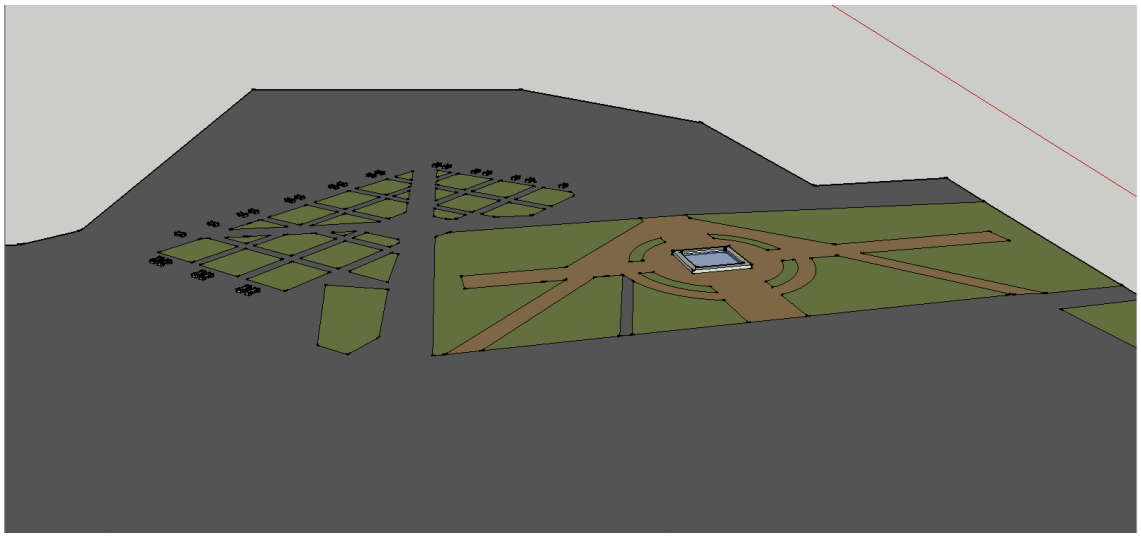


Figura 36: Vista desde el oeste del suelo simulado del campus de la EPS

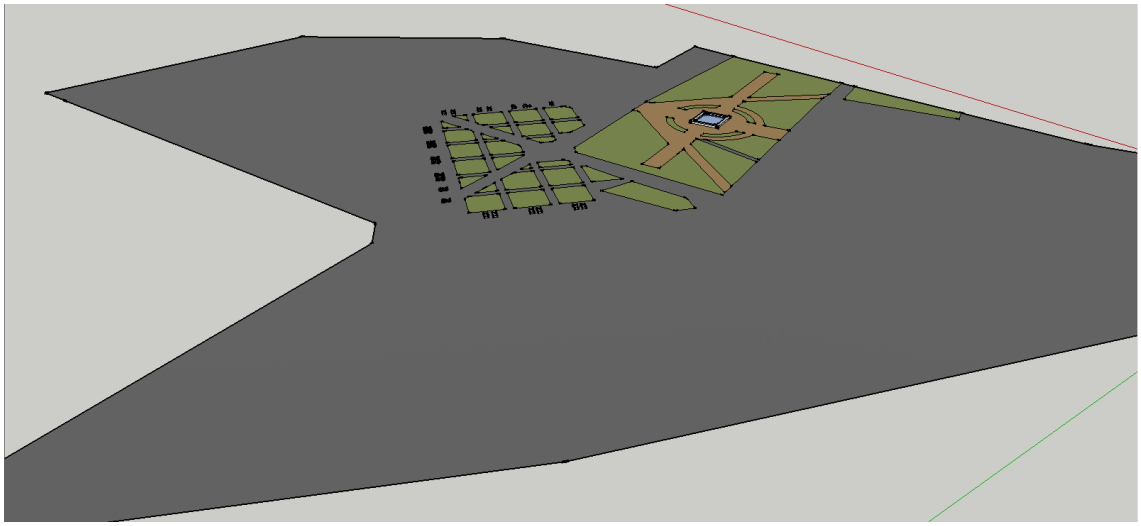


Figura 37: Vista desde el norte del suelo simulado del campus de la EPS

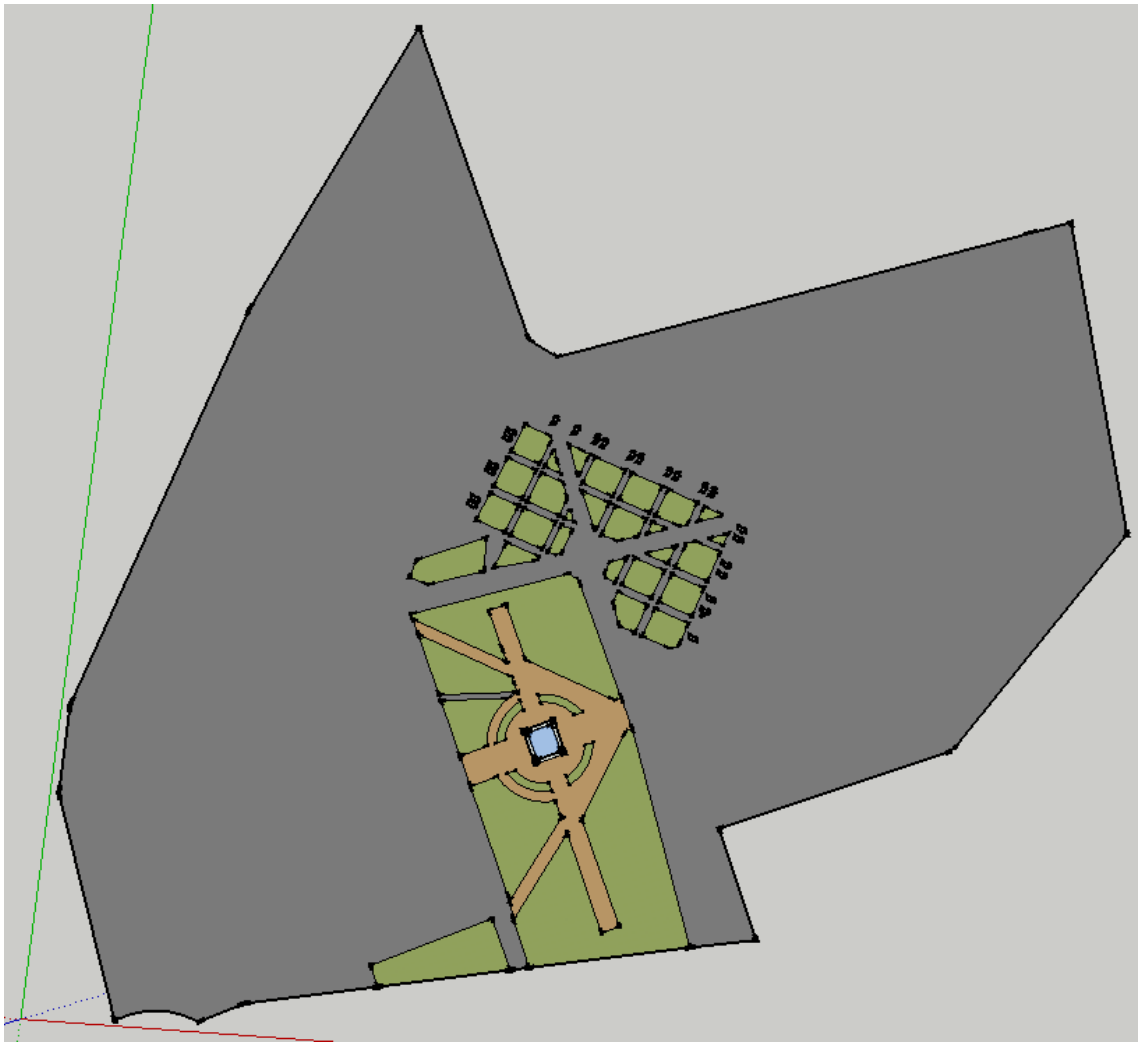


Figura 38: Vista cenital del suelo simulado del campus de la EPS

4.2. Webots

El software utilizado para llevar a cabo la construcción del entorno simulado final y el modelado del vehículo autónomo, así como la ejecución del controlador de dicho vehículo ha sido Webots.

Las simulaciones en Webots, también llamadas "mundos", están definidas por un árbol de nodos. Cada nodo dentro del árbol tiene una serie de propiedades personalizables o "campos". Los mundos de Webots se guardan en archivos con el sufijo .wbt, cuyo formato deriva del lenguaje VRML.

Los nodos de Webots que forman el mundo se organizan en una estructura de árbol llamada "árbol de escena". Este árbol de escena se muestra al usuario de dos formas: en el centro de la ventana del programa se muestra la representación en tres dimensiones del árbol de escena, y en la parte izquierda de la ventana del programa se muestra la representación jerárquica del árbol de escena.

Figura 39: Ejemplo de mundo simulado en Webots

Una vez modeladas las simulaciones de los edificios del campus y el suelo mediante SketchUp, éstas se han exportado en forma de archivos VRML para, gracias a la funcionalidad que ofrece Webots de importar este tipo de archivos, introducirlas en el entorno simulado. Una vez se ha introducido un objeto en Webots mediante su archivo VRML, este se ha exportado de nuevo en forma de objeto de Webots (archivo con extensión .wbo) para poder trabajar de una forma más cómoda con él dentro del programa, ya que estos archivos se pueden introducir directamente como nodos de Webots.

Además, Webots ofrece amplias opciones concretas para el diseño de robots móviles, pudiendo usar nodos con diferentes formas geométricas y tamaños para construir la estructura del robot y gran variedad de dispositivos como GPS, motores, láseres o cámaras (ver Figura 40). Estas herramientas de modelado que ofrece Webots se han usado para construir el vehículo autónomo dentro del propio programa para poder usarlo en la simulación junto con el modelado del campus.

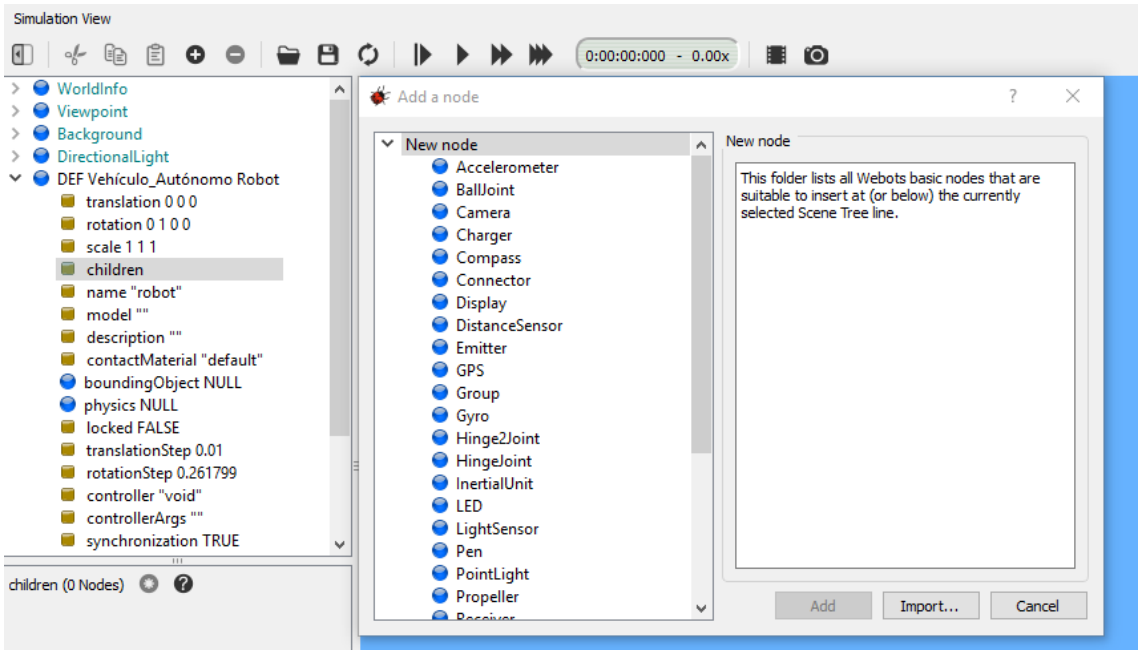


Figura 40: Elementos para crear un robot en Webots

4.2.1. Campus de la EPS Carlos III

Una vez realizado el proceso de importación y exportación de los elementos modelados mediante SketchUp, se han combinado todos dentro de Webots para crear el entorno simulado final por que el circulará el vehículo autónomo simulado. A continuación, se pueden observar imágenes del entorno simulado final:

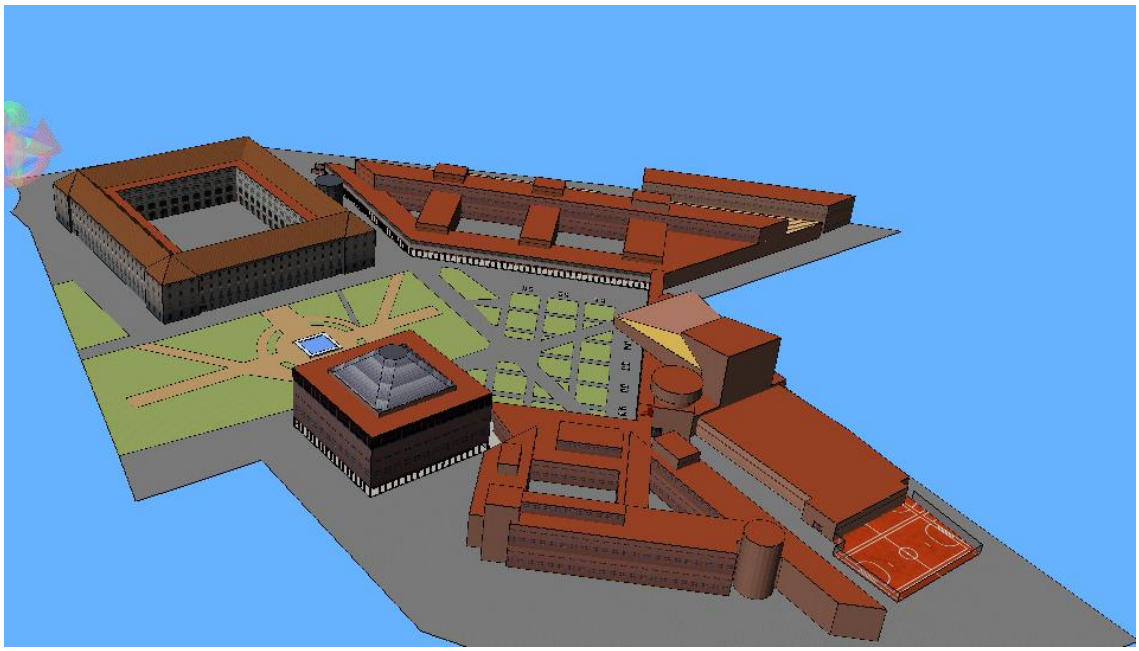


Figura 41: Vista desde el oeste del campus simulado de la EPS en Webots

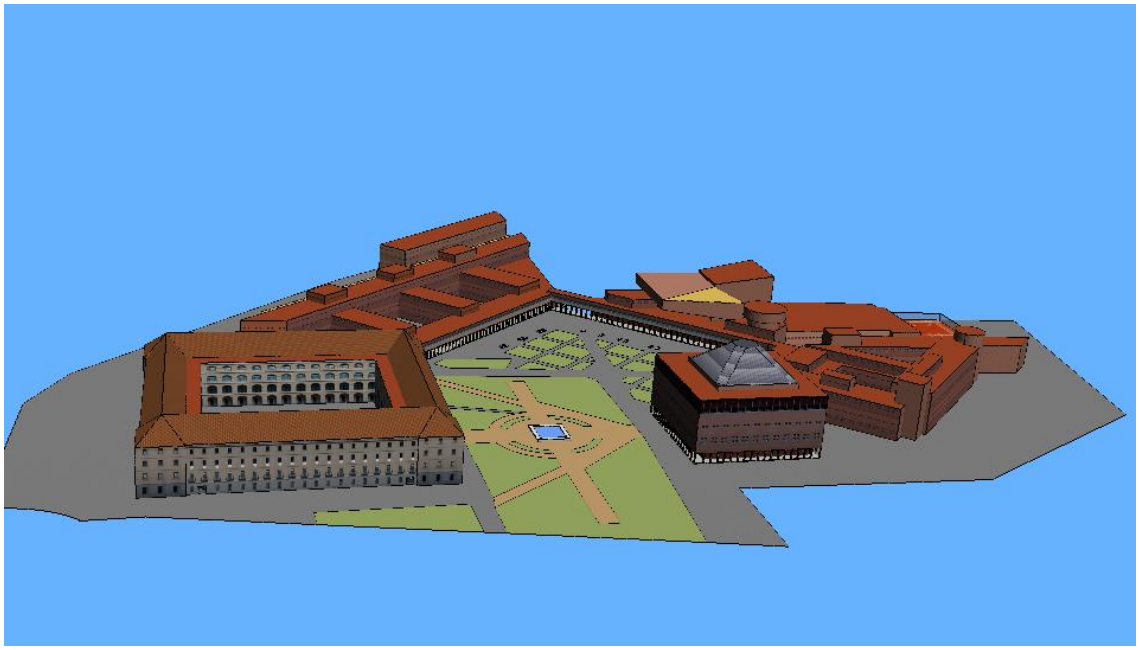


Figura 42: Vista desde el sur del campus simulado de la EPS en Webots

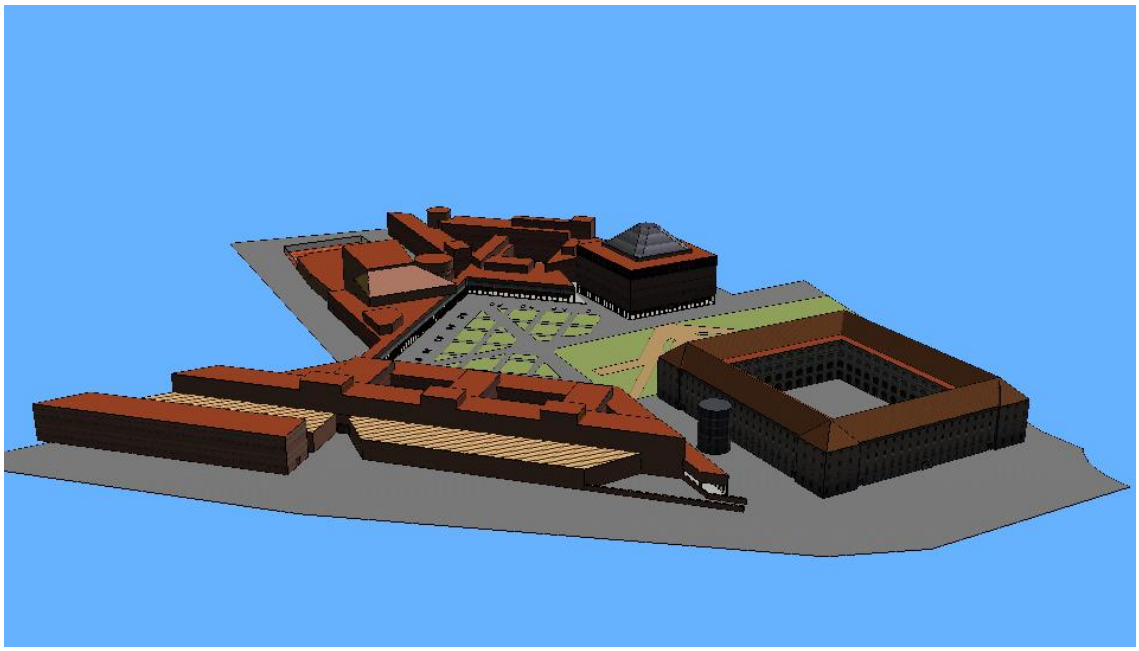


Figura 43: Vista desde el este del campus simulado de la EPS en Webots

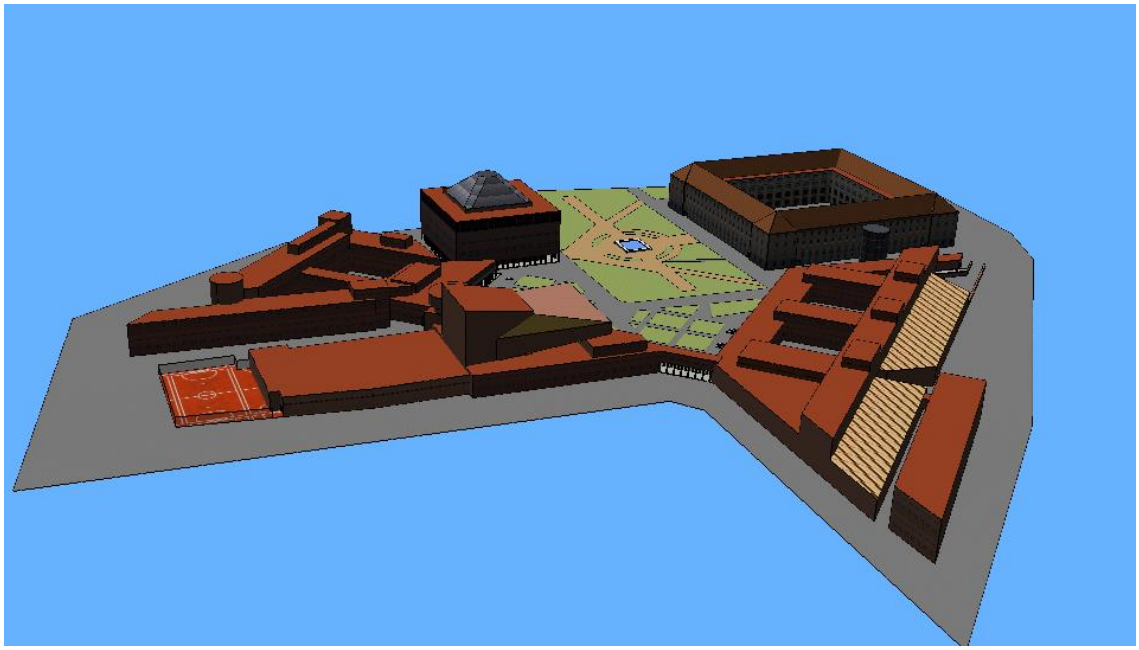


Figura 44: Vista desde el norte del campus simulado de la EPS en Webots

Figura 45: Vista cenital del campus simulado de la EPS en Webots

Debido a la carga en memoria que supone simular todos los edificios del campus, así como el suelo, la versión de Webots utilizada en este proyecto tiene problemas para ejecutar la simulación en el entorno simulado con este nivel de detalle, por lo que se ha desarrollado otra versión del campus con unos edificios sin detalle, pero manteniendo su forma básica y el suelo para poder realizar las pruebas del controlador del vehículo autónomo. En esta versión, también se ha añadido la expansión más reciente del edificio Juan Benet:

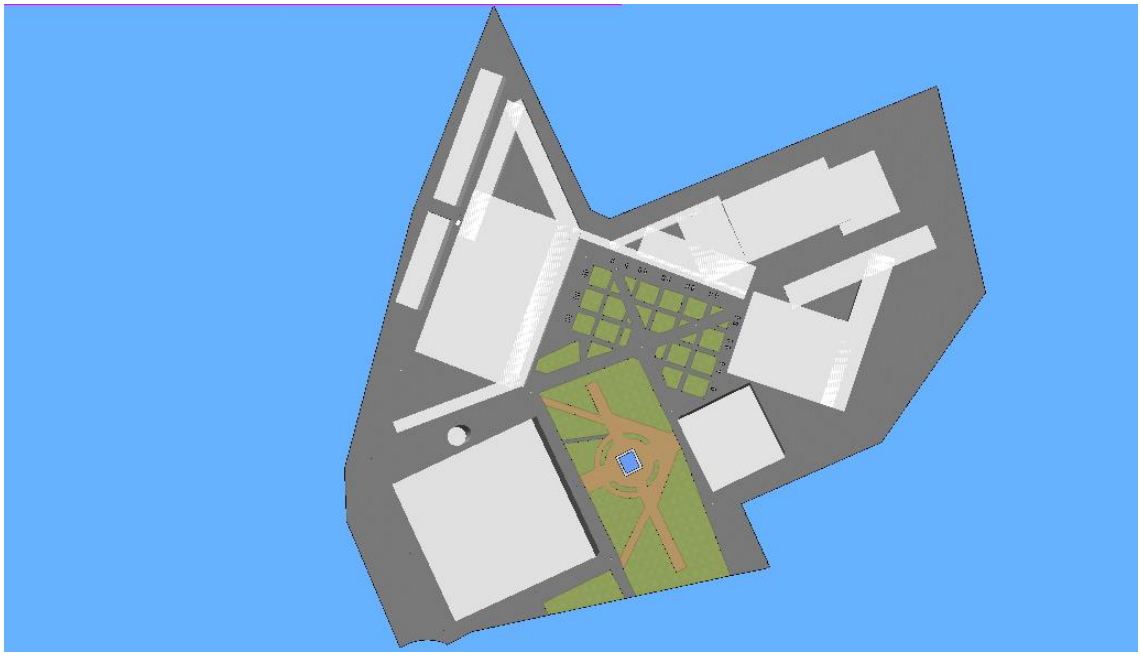


Figura 46: Vista cenital del campus simulado básico de la EPS en Webots

En la imagen siguiente se puede observar una comparativa entre el campus real, la simulación detallada y la simulación básica:

Figura 47: Comparativa entre campus detallado, campus real y campus básico en Webots

4.2.2. Vehículo autónomo

El vehículo autónomo que circulará por el campus anteriormente descrito es una simulación de un vehículo real propiedad del Laboratorio de Sistemas Inteligentes (LSI) de la Universidad Carlos III de Madrid. El vehículo que se ha simulado es un coche de golf de la compañía EZ-GO, más concretamente el modelo TXT [52], modificado por el LSI para añadirle un módulo de conducción automática, así como cámaras y sensores.

Figura 48: Vehículo autónomo físico

Al vehículo se le han realizado modificaciones en los sistemas mecánico y eléctrico. De acuerdo con esto, el volante se ha retirado para instalar el sistema motor-codificador para controlar la dirección del vehículo de forma electrónica. De forma adicional, el pedal del acelerador se ha desactivado para controlar la tracción eléctrica del motor de movimiento mediante un amplificador de potencia y dirigido por un microcontrolador PIC. Las entradas del microcontrolador son el porcentaje de la capacidad máxima para el estator, rotor, y el ángulo deseado de las ruedas. El vehículo cuenta además con un GPS y una unidad de medición inercial (IMU por sus siglas en inglés) situados en el centro del vehículo a 170cm de altura (ver Figura 49).

Figura 49: Lateral vehículo autónomo físico

El coche de golf cuenta en su parte frontal con una cámara binocular de visión estéreo modelo Bumblebee 2 [53]. La cámara tiene una resolución máxima de 1032x776 pixeles a 20 frames por segundo [54]. Está montada sobre el parabrisas delantero unos 160 cm sobre el suelo y orientada a -45 grados. La cámara cumple tres propósitos: primero construir un mapa libre del trazado para navegar por el entorno, seguido de una odometría visual y finalmente la detección de obstáculos o peatones (ver Figura 50).

Figura 50: Cámara Bumblebee 2 en vehículo autónomo físico

El vehículo cuenta también con un láser telémetro de la marca SICK (modelo LMS 291) [55]. El dispositivo cuenta con un rango de escaneo de 180 grados con 0.25 grados de resolución angular [56]. Está montado en el parachoques frontal del vehículo a unos 30 cm sobre el suelo. Para evitar la detección de las ruedas delanteras, el rango de escaneo está limitado a 100 grados a 20Hz (ver Figura 51)

Figura 51: Laser SICK LMS 291 en vehículo autónomo físico

El modelado del vehículo autónomo, a diferencia del campus, se ha realizado íntegramente con las herramientas de modelado que proporciona Webots. Se ha hecho un modelado muy básico, respetando en la medida de lo posible las dimensiones del vehículo real y la disposición de los elementos externos, debido a que, si se le añade una mayor complejidad al modelado, la memoria máxima de que dispone el programa no es capaz de ejecutar la simulación de una forma fluida. A continuación, se muestra el resultado de la simulación del vehículo autónomo:

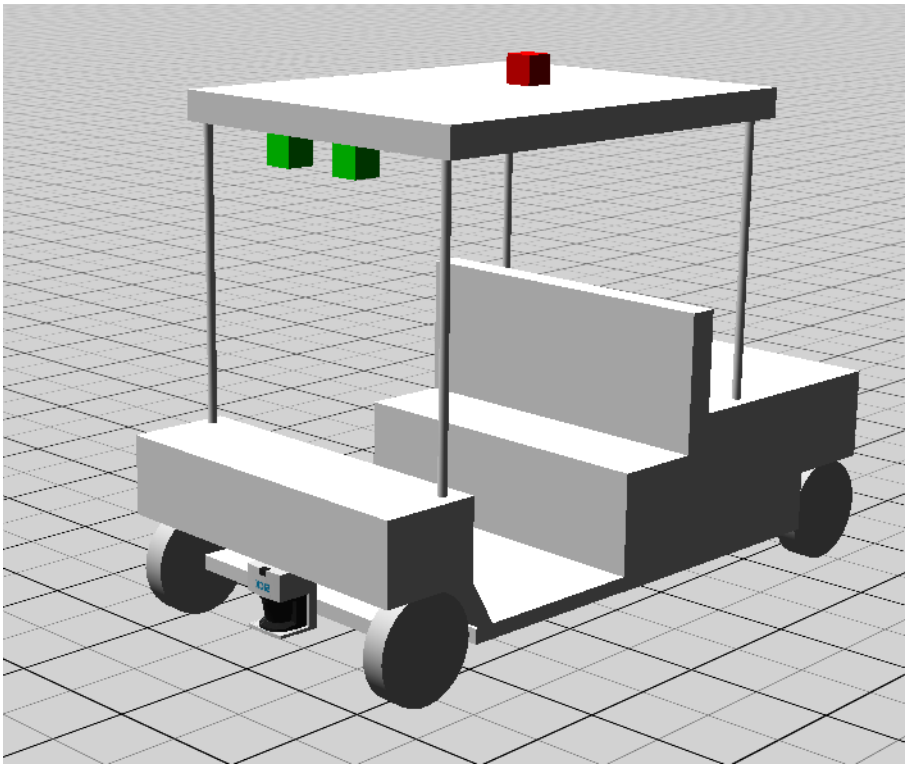


Figura 52. Vehículo autónomo simulado en Webots

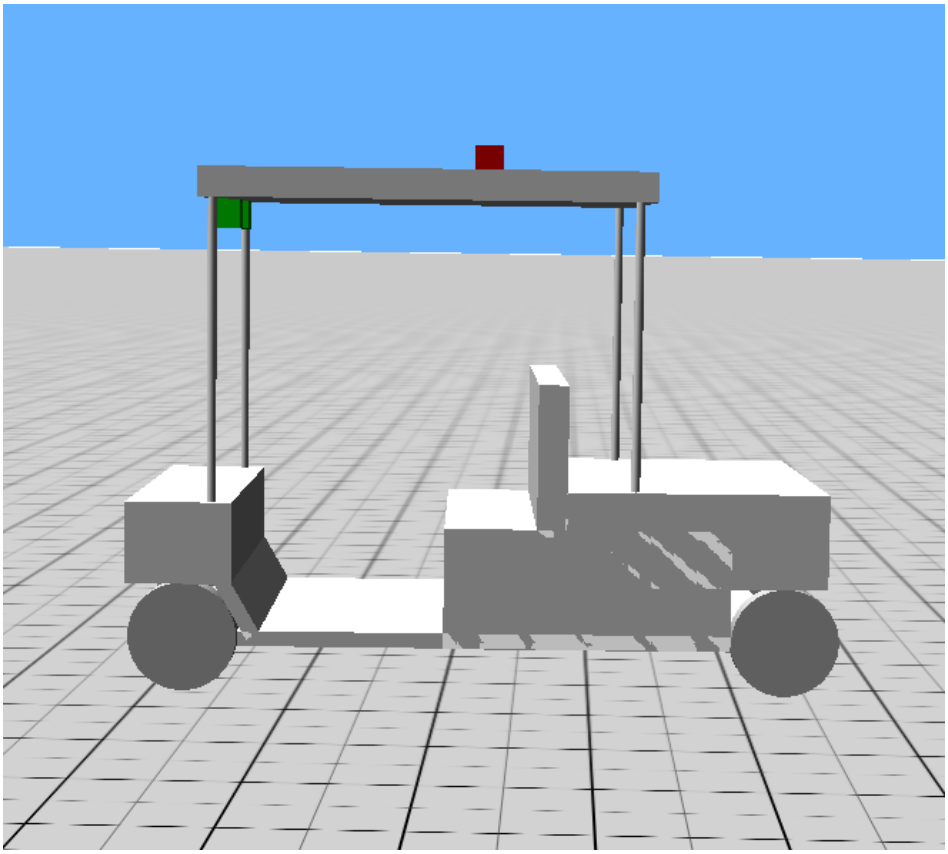


Figura 53: Lateral del vehículo autónomo simulado en Webots

Para simular la cámara estéreo Bumblebee 2, se han instalado dos cámaras independientes en la parte correspondiente a dicha cámara en el vehículo físico. Además, se ha añadido un GPS en el techo del vehículo (ver Figura 54).

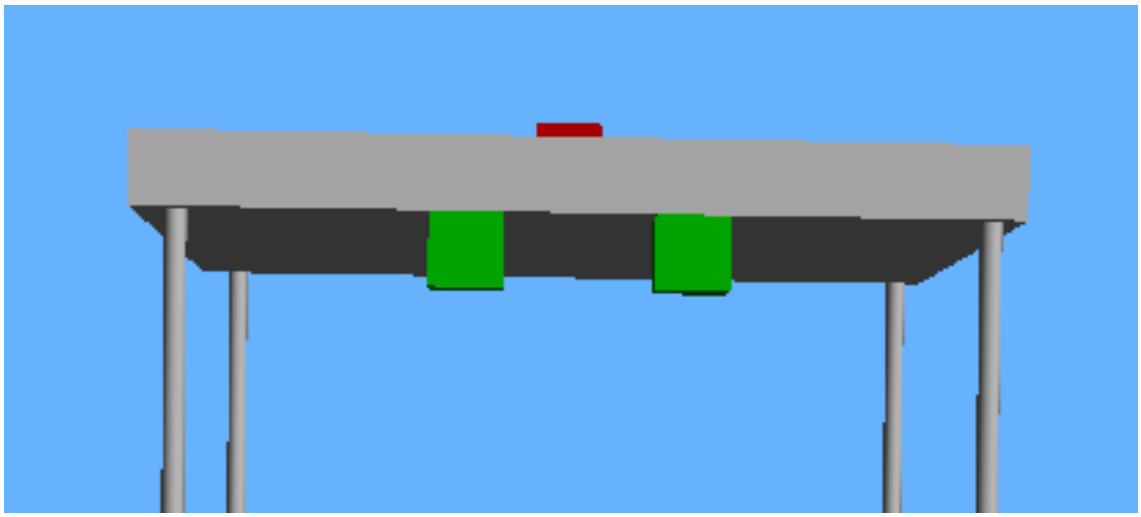


Figura 54: Cámaras en vehículo autónomo simulado en Webots

Para simular el laser SICK (LMS 291) se ha utilizado el modelo de laser SICK que ya existe en Webots y que coincide con el modelo que monta el vehículo físico (ver Figura 55)

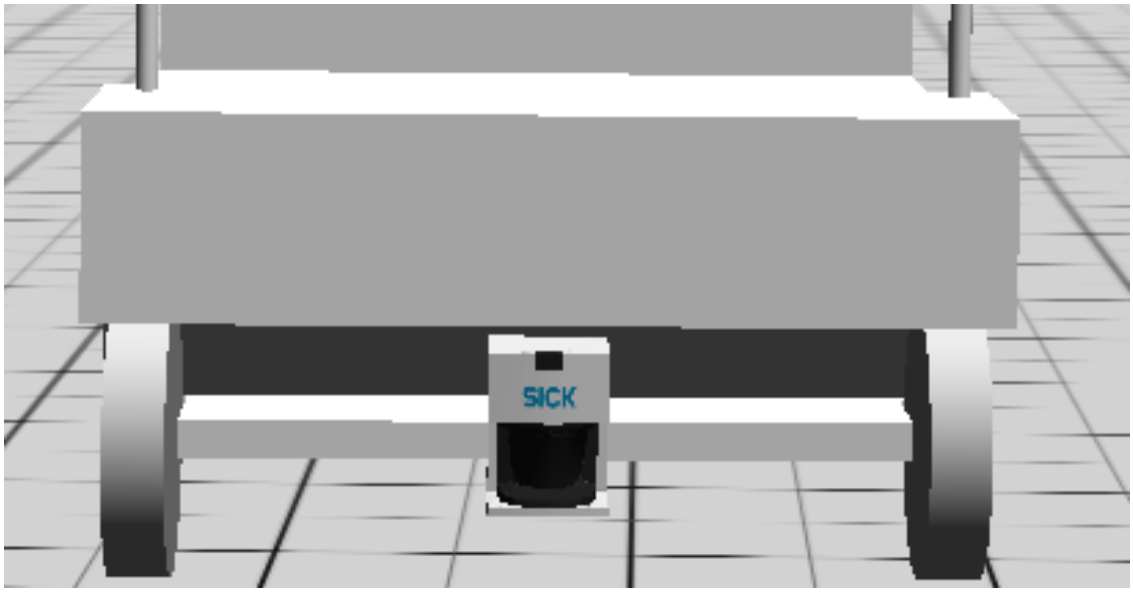


Figura 55: Láser SICK LMS 291 en vehículo autónomo simulado en Webots

Además de estos dispistivos, las ruedas delanteras del vehículo se han definido como motores, tanto motrices como directrices, por lo que estas ruedas serán las que dirijan el movimiento del vehículo durante la simulación. También se ha añadido un dispositivo brújula (invisible) para conocer la dirección en la que está orientado el vehículo para así poder calcular durante la simulación cual es la dirección que tiene que seguir para alcanzar su objetivo.

En la Figura 56 puede observarse una comparativa del carro real y el carro que se ha modelado para la simulación.

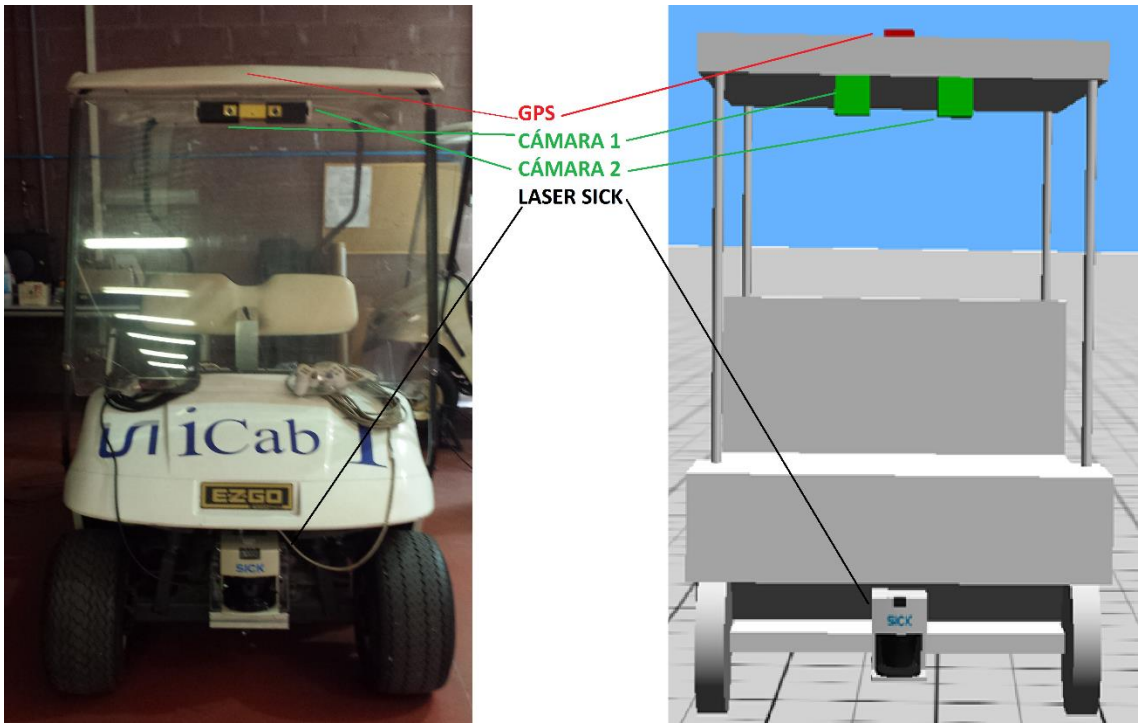


Figura 56: Comparativa entre vehículo autónomo físico y simulado en Webots

CAPÍTULO 5
Implementación del Sistema

5. Implementación del Sistema

En esta sección se describe la funcionalidad del controlador, así como la implementación realizada de éste para el vehículo autónomo simulado. Todo el código realizado ha sido implementado en el lenguaje de programación C++.

5.1. Funcionalidad del controlador

La funcionalidad desarrollada del controlador es la de llevar el vehículo autónomo desde un punto inicial dentro del campus simulado hasta otro punto dentro del campus proporcionado por el usuario. Para llevar a cabo esta funcionalidad, se ha desarrollado la siguiente solución:

1. Se establecen distintos puntos de ruta y puntos de destino dentro del entorno simulado.
2. Se coloca el vehículo autónomo en una posición inicial dentro del entorno simulado.
3. El controlador, teniendo en cuenta la posición inicial del vehículo y la posición de destino, mediante un algoritmo de búsqueda elabora una ruta por los puntos de ruta definidos anteriormente para llegar al punto de destino.
4. Una vez establecida la ruta, el controlador pone en marcha el vehículo autónomo y recorre uno a uno los puntos de ruta establecidos.
5. Si mediante el dispositivo laser SICK del vehículo se detecta un obstáculo en la trayectoria, el controlador abandona la tarea de circular por los puntos de ruta y procede a evitar el obstáculo. Una vez evitado, el controlador continúa con el recorrido del vehículo por los puntos de ruta.
6. Cuando el vehículo alcanza el punto de destino, el controlador detiene el vehículo.

Esta solución se ajusta correctamente tanto a la funcionalidad requerida (ir de un punto a otro dentro de la simulación) como a los elementos de los que se dispone (el vehículo autónomo y sus dispositivos). Pueden realizarse cambios en elementos como la posición de los puntos de ruta, la cantidad de puntos de ruta o el algoritmo de búsqueda de caminos escogido, pero el desarrollo principal de la funcionalidad será el mismo.

5.2. Implementación en C++

Como se ha comentado anteriormente, se ha elegido el lenguaje de programación C++ para la implementación del controlador del vehículo autónomo. Se ha optado por este lenguaje debido a que los comandos necesarios para la posterior implementación de ROS pueden realizarse con este lenguaje y dentro de la amplia oferta de lenguajes que ofrecía WEBOTS era la mejor opción para el programador.

A continuación, se muestra el diagrama de clases del proyecto (ver Figura 57):

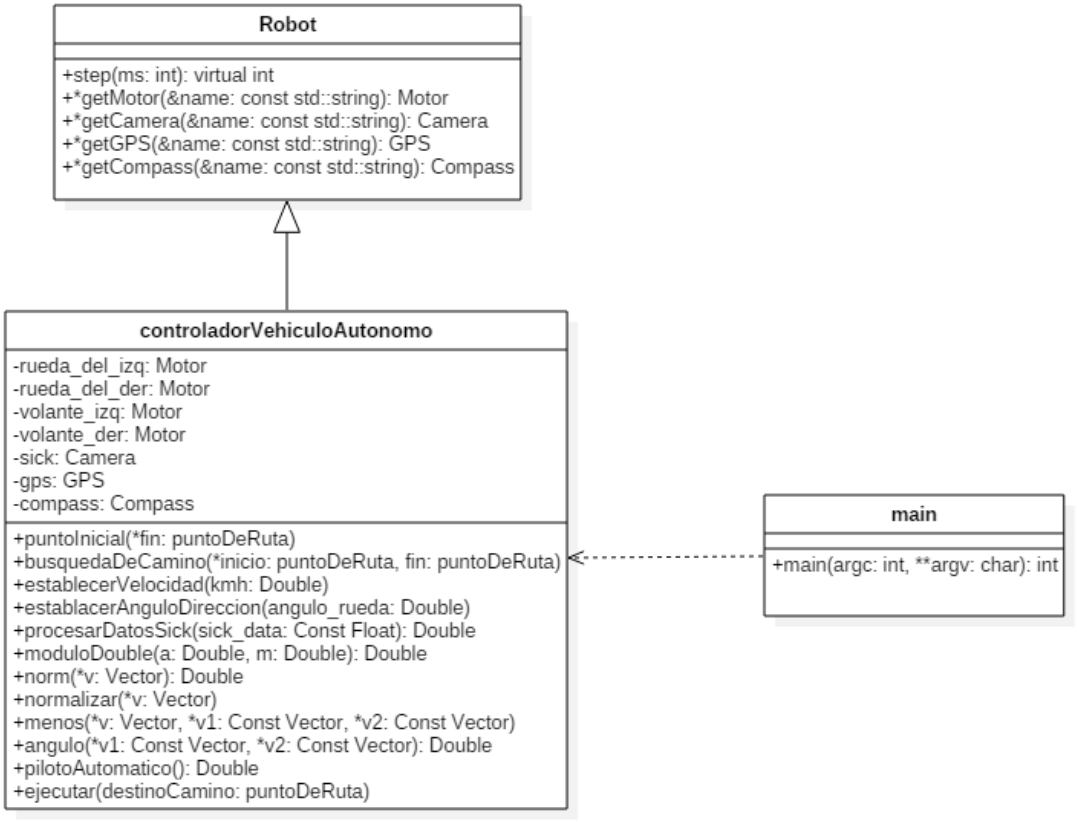


Figura 57: Diagrama de clases

Una vez visto el diagrama de clases, en los apartados siguientes se describen todas las clases, funciones y atributos del proyecto.

5.2.1. Clase Robot

La clase `Robot` es una clase propia de Webots y se encarga de definir los atributos y funciones comunes a cualquier objeto de tipo "Robot" que puede crearse dentro del programa (en este caso, el vehículo autónomo se ha modelado como un robot). La clase `Robot` tiene multitud de atributos y funciones, pero tanto en el diagrama de clases como en este apartado solo se reflejan aquellos que son utilizados por el controlador creado para el vehículo autónomo.

5.2.1.1. Funciones

Las funciones de esta clase son las siguientes:

- **`step`**: esta función tiene como objetivo avanzar tantos instantes de tiempo como se le indique en milisegundos.
    - **Entrada**: tiempo en milisegundos a avanzar.
    - **Salida**: se avanzan los milisegundos requeridos en la simulación.

- **`getCamera`:** esta función tiene como objetivo obtener el puntero a un dispositivo cámara del robot mediante el nombre que tiene ésta dentro de la simulación.
    - **Entrada:** nombre del dispositivo cámara dentro de la simulación.
    - **Salida:** puntero al dispositivo cámara cuyo nombre se corresponde con el nombre introducido.

- **`getCompass`:** esta función tiene como objetivo obtener el puntero a un dispositivo brújula del robot mediante el nombre que tiene éste dentro de la simulación.
    - **Entrada:** nombre del dispositivo brújula dentro de la simulación.
    - **Salida:** puntero al dispositivo brújula cuyo nombre se corresponde con el nombre introducido.

- **`getGPS`:** esta función tiene como objetivo obtener el puntero a un dispositivo GPS del robot mediante el nombre que tiene éste dentro de la simulación.
    - **Entrada:** nombre del dispositivo GPS dentro de la simulación.
    - **Salida:** puntero al dispositivo GPS cuyo nombre se corresponde con el nombre introducido.

- **`getMotor`:** esta función tiene como objetivo obtener el puntero a un dispositivo motor del robot mediante el nombre que tiene éste dentro de la simulación.
    - **Entrada:** nombre del dispositivo motor dentro de la simulación.
    - **Salida:** puntero al dispositivo motor cuyo nombre se corresponder con el nombre introducido.

5.2.2. Clase controladorVehiculoAutonomo

Esta es la clase principal del proyecto y define como inicializar y ejecutar el controlador. Contiene todos los métodos encargados de que el vehículo autónomo circule desde un punto a otro punto dado dentro de la simulación de manera autónoma y evitando los obstáculos que se presenten. Esta clase deriva de la clase Robot de Webots y por tanto hereda todas sus funciones.

5.2.2.1. Atributos

Los atributos que forma parte de esta clase son:

- **`rueda_del_izq`:** puntero que apunta al dispositivo motor de la rueda delantera izquierda del vehículo autónomo.
- **`rueda_del_der`:** puntero que apunta al dispositivo motor de la rueda delantera derecha del vehículo autónomo.
- **`volante_izq`:** puntero que apunta al dispositivo motor de dirección de la rueda delantera izquierda del vehículo autónomo.
- **`volante_der`:** puntero que apunta al dispositivo motor de dirección de la rueda delantera derecha del vehículo autónomo.
- **`sick`:** puntero que apunta al dispositivo laser SICK del vehículo autónomo.

- **gps:** puntero que apunta al dispositivo GPS del vehículo autónomo.
- **compass:** puntero que apunta al dispositivo brújula del vehículo autónomo.

5.2.2.2. Funciones

Las funciones de esta clase son las siguientes:

- **puntoInicial:** esta función tiene como objetivo calcular cual es el primer punto de ruta al que se tiene que dirigir el vehículo autónomo teniendo en cuenta su posición inicial.
    - **Entrada:** puntero al punto de ruta final.
    - **Salida:** se introduce el punto de ruta objetivo en la lista de puntos a recorrer.

- **busquedaDeCamino:** esta función tiene como objetivo obtener el camino de puntos de ruta que debe seguir el vehículo autónomo desde el punto inicial hasta el punto de ruta definido como destino.
    - **Entrada:** puntero al punto de ruta inicial, puntero al punto de ruta final.
    - **Salida:** se introducen uno a uno los puntos de ruta por los que debe circular el vehículo en la lista de puntos de ruta.

- **establecerVelocidad:** esta función tiene como objetivo establecer la velocidad a la que debe circular el vehículo autónomo.
    - **Entrada:** velocidad deseada en kilómetros por hora.
    - **Salida:** se establece la velocidad de cada rueda motriz del vehículo autónomo a la velocidad deseada transformada a radianes por segundo.

- **establecerAnguloDireccion:** esta función tiene como objetivo establecer el ángulo de giro de las ruedas.
    - **Entrada:** ángulo de giro deseado.
    - **Salida:** se establece el ángulo de giro de las ruedas directrices del vehículo autónomo. Si el ángulo es positivo el giro es hacia la derecha y viceversa si el ángulo es negativo.

- **procesarDatosSick:** esta función tiene como objetivo procesar los datos recogidos por el láser SICK para comprobar si se detecta algún obstáculo y si es así, calcular el ángulo al que está dicho obstáculo respecto al vehículo autónomo.
    - **Entrada:** puntero a los datos recogidos por el láser SICK.
    - **Salida:** si no se detecta obstáculo se retorna un valor predefinido. Si se detecta un obstáculo se retorna el ángulo aproximado del obstáculo respecto al vehículo.

- **moduloDouble:** esta función tiene como objetivo calcular el módulo de la división entre dos números dados.
    - **Entrada:** dos números cuyo módulo se va a calcular.
    - **Salida:** se retorna el valor del módulo.

- **norm:** esta función tiene como objetivo calcular la norma de un vector dado.
    - **Entrada:** puntero al vector cuya norma se quiere calcular.
    - **Salida:** se retorna el valor de la norma del vector.

- **normalizar:** esta función tiene como objetivo normalizar un vector dado.
    - **Entrada:** puntero al vector que se quiere normalizar.
    - **Salida:** se retorna el vector normalizado.

- **menos:** esta función tiene como objetivo ejecutar la resta de dos vectores dados y guardarla en un tercer vector.
    - **Entrada:** puntero al vector donde se va a almacenar el resultado de la resta, punteros a los dos vectores a restar.
    - **Salida:** se almacena en el vector dado el resultado de la resta de vectores.

- **angulo:** esta función tiene como objetivo calcular el ángulo entre dos vectores dados.
    - **Entrada:** punteros a los dos vectores cuyo ángulo se quiere calcular.
    - **Salida:** se retorna el valor del ángulo formado por los dos vectores.

- **pilotoAutomatico:** esta función tiene como objetivo llevar a cabo la funcionalidad principal del controlador. Es el encargado de que el vehículo recorra una a una las posiciones objetivo, ya que calcula cual debe ser el giro que debe realizar el vehículo autónomo para dirigirse a su destino actual.
    - **Entrada:** ninguna.
    - **Salida:** si no quedan objetivos que alcanzar (se ha llegado al destino) se retorna como valor de giro el valor 0. Si hay un objetivo a alcanzar actualmente activo y la distancia hasta dicho objetivo es suficientemente pequeña como para considerarlo alcanzado, se elimina ese objetivo de la lista y se retorna como valor de giro el valor 0. Si no se da ninguno de los casos anteriores se calcula si el giro debe ser hacia la izquierda o hacia la derecha. Si el giro debe ser hacia la derecha se retorna como valor de giro el valor 1, y si el giro debe ser hacia la izquierda se retorna como valor de giro el valor -1.

- **`ejecutar`:** esta función tiene como objetivo llevar a cabo la ejecución del controlador. Se encarga de llevar a cabo el flujo de la simulación, ejecutando cuando sea necesario las funciones descritas anteriormente mientras dure la simulación, es decir, hasta que se detenga el programa o se alcance la posición objetivo.
    - **Entrada:** punto de ruta objetivo que se desea alcanzar.
    - **Salida:** el vehículo autónomo circula desde la posición inicial hasta la posición destino dada como entrada recorriendo una serie de puntos de ruta y esquivando obstáculos cuando se presentan. Al alcanzar la posición objetivo, el vehículo se detiene.

5.2.3. Función main

La función `main` es la función principal del controlador. En ella, se crea una instancia del controlador que se ha definido (la clase `controladorVehículoAutonomo` que se ha detallado anteriormente) y se lanza su función principal "`ejecutar`" y se destruye al finalizar la ejecución. Hay que tener en cuenta que solo se puede crear una instancia del controlador a la vez.

5.3. ROS

Antes de exponer la implementación realizada del controlador del vehículo autónomo mediante ROS, se va a realizar una introducción a este marco de trabajo, su funcionamiento, y porque se ha decidido desarrollar el controlador con él.

5.3.1. ¿Qué es ROS?

El Sistema Operativo Robótico, o ROS [57] por sus siglas en Inglés (*Robotic Operating System*) es un marco de trabajo flexible y gratuito para desarrollar código para robots. Se trata de una colección de herramientas, librerías y convenios que buscan simplificar las tareas que suponen crear comportamientos robóticos complejos y robustos a través de una amplia variedad de plataformas robóticas.

ROS es necesario porque crear software robótico de propósito general que sea verdaderamente robusto es complicado. Desde la perspectiva de un robot, problemas que parece triviales para aun ser humano normalmente pueden variar enormemente entre instancias de tareas y entornos. Manejar estas variaciones es tan complejo que ningún usuario, laboratorio, o institución puede aspirar a hacerlo por su cuenta.

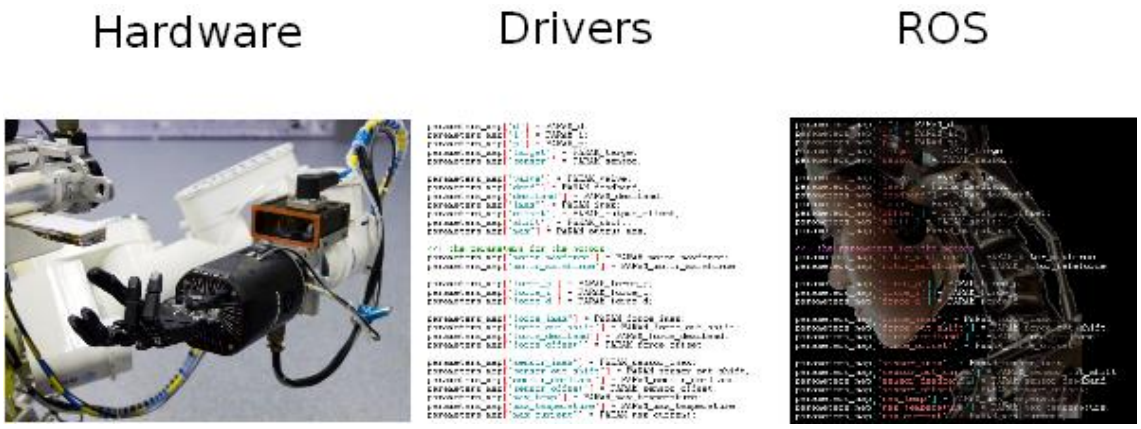


Figura 58: Sistema Robótico Operativo ROS [57]

Como resultado, ROS fue desarrollado desde cero para fomentar el desarrollo colaborativo de software robótico. Por ejemplo, un laboratorio puede tener expertos en mapeado de interiores y contribuir con un sistema de primer nivel para realizar mapas, mientras que otro grupo puede ser especialista en usar mapas para navegación y otro diferente puede haber desarrollado una aproximación de visión computarizada capaz de reconocer de forma correcta objetos pequeños desordenados. ROS ha sido diseñado específicamente para que grupos como los mencionados anteriormente colaboren y construyan su trabajo con la ayuda del trabajo de otros.

5.3.2. ¿Cómo funciona ROS?

Los elementos principales que forman parte del funcionamiento de ROS son los siguientes:

- **Gráfico de Computación:** el Gráfico de Computación es la red de pares de los procesos de ROS que están procesando datos conjuntamente.
- **Nodos:** los nodos son procesos que realizan un cómputo. ROS está diseñado para ser modular; un sistema de control de un robot normalmente comprende muchos nodos.
- **Nodo Maestro:** el Nodo Maestro de ROS proporciona el registro de nombres y búsqueda al resto del Gráfico de Computación.
- **Mensajes:** los nodos se comunican unos con otros mediante mensajes. Un mensaje es una estructura de datos simple, que comprende diferentes campos de tipos primitivos de datos (enteros, con coma flotante, booleanos, etc.).
- **Hilos:** los mensajes son enrutados mediante un sistema de transporte que utiliza una semántica de publicación/subscripción. Un nodo envía un mensaje mediante su publicación en un hilo dado. El hilo es un nombre que se usa para identificar el contenido del mensaje. Un nodo que esté interesado en un dato en concreto, se suscribirá al hilo apropiado para conseguirlo. Puede haber múltiples publicadores y subscriptores para un mismo hilo, y un mismo nodo puede publicar o subscribirse a varios hilos diferentes. Normalmente, los publicadores y suscriptores no son conscientes de la existencia de los demás.
- **Servicios:** el modelo de publicación/suscripción es un paradigma de comunicación muy flexible, pero su transporte muchos-a-muchos en un solo sentido no es apropiado para

las interacciones de tipo petición/respuesta que se requieren en ocasiones en un sistema distribuido. Estas interacciones son realizadas mediante los servicios, que están definidos por un par de estructuras de mensajes: una para la petición y otra para la respuesta. Un nodo ofrece un servicio mediante un nombre y un cliente usa el servicio enviándole el mensaje de petición y esperando la respuesta.

El primer paso para ejecutar un programa en ROS es siempre inicializar el Nodo Maestro, ya que éste se encarga de guardar la información de registro de los hilos y servicios de los nodos de ROS y de mantener esta información actualizada si hay algún cambio. Una vez inicializado el Nodo Maestro, se pueden ejecutar los nodos que forman el programa a ejecutar. Estos nodos se comunican entre ellos mediante hilos y servicios hasta que termina la ejecución del programa.

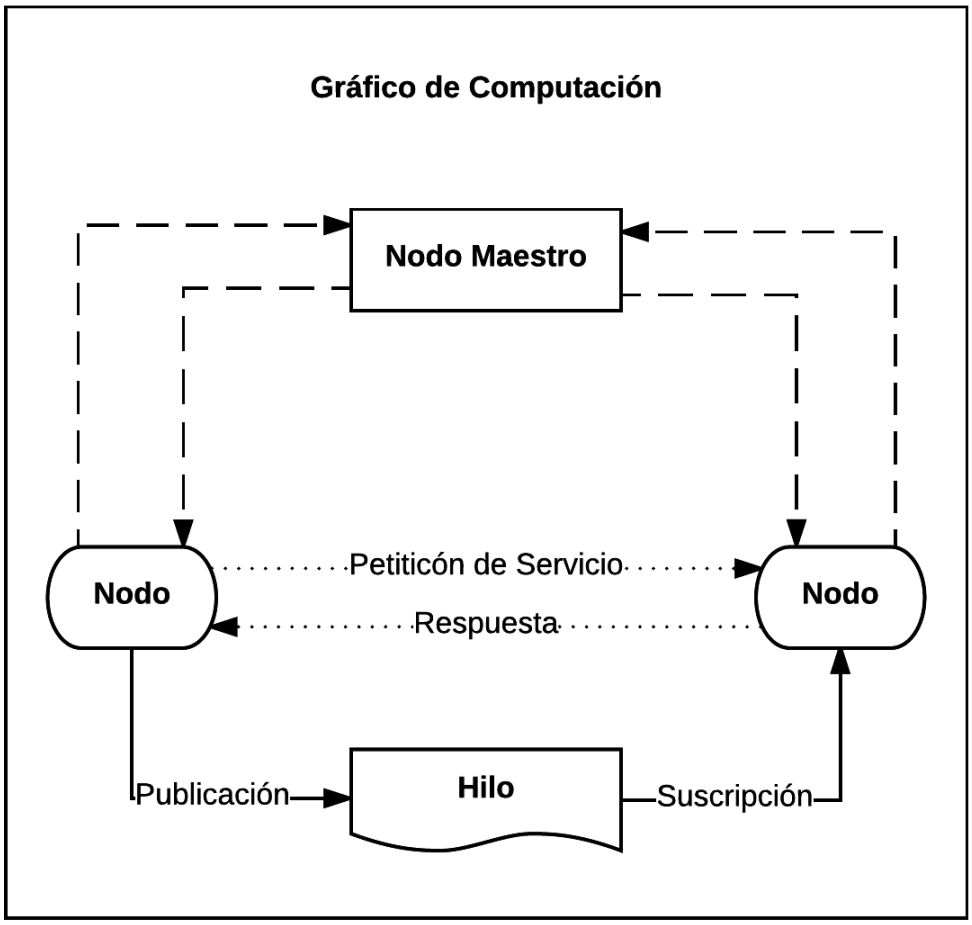


Figura 59: Esquema del Gráfico de Computación de ROS

5.3.3. ¿Por qué usar ROS?

Se ha decidido implementar el controlador del vehículo autónomo mediante el uso de ROS debido a la adaptabilidad que ofrece este marco de trabajo. Al implementar el código mediante ROS, el mismo controlador usado para operar el vehículo autónomo simulado dentro de Webots, puede servir en un futuro para operar un vehículo autónomo real sin tener que modificar en gran medida el funcionamiento del controlador.

Además, al adaptarse a una serie de normas y convenios comunes de funcionamiento, este trabajo puede ser retomado por otros investigadores o incluso realizar un trabajo en paralelo con el controlador implementado de una forma sencilla.

5.3.4. Implementación con ROS

La implementación del controlador mediante ROS es muy similar a la explicada anteriormente en C++. El lenguaje de programación utilizado es el mismo, así como el código principal del controlador. Las principales diferencias entre ambas implementaciones están en el uso de los dispositivos del vehículo en el controlador principal y en que el controlador que usará directamente el vehículo en Webots será un controlador de ROS predefinido que se comunicará mediante hilos con el controlador creado para el vehículo. A continuación, se detallan en estas diferencias.

5.3.4.1. Funciones de ROS

Para habilitar la comunicación entre el controlador y el vehículo autónomo dentro de Webots, se han añadido una serie de funciones propias de ROS y predefinidas en el entorno del programa. A continuación, se detallan estas funciones.

- **`controllerNameCallback`:** esta función tiene como objetivo rellenar una lista con los nombres de los controladores disponibles en la red de ROS (en este caso solo habrá uno, el controlador de ROS que tiene el vehículo autónomo).
  - **Entrada:** mensaje con el nombre del controlador.
  - **Salida:** se introduce el controlador en la lista de controladores y aumenta el contador de controladores.

- **`cameraRangeFinderCallback`:** esta función tiene como objetivo almacenar la imagen captada por el dispositivo láser SICK.
  - **Entrada:** valores recogidos por el dispositivo láser SICK.
  - **Salida:** los valores recogidos por el dispositivo láser SICK quedan almacenados en su variable correspondiente.

- **`GPSCallback`:** esta función tiene como objetivo almacenar la posición captada por el dispositivo GPS.
  - **Entrada:** valores recogidos por el dispositivo GPS.
  - **Salida:** los valores recogidos por el dispositivo GPS quedan almacenados en su variable correspondiente.

- **`compassCallback`:** esta función tiene como objetivo almacenar los valores recogidos por el dispositivo brújula.
  - **Entrada:** valores recogidos por el dispositivo brújula.
  - **Salida:** los valores recogidos por el dispositivo brújula quedan almacenados en su variable correspondiente.

- **`quit`:** esta función tiene como objetivo detener la ejecución del controlador.
    - **Entrada:** señal de finalización.
    - **Salida:** se detiene la simulación y termina con la comunicación entre el controlador del vehículo y el hilo de ROS.

5.3.4.2. Controlador Vehículo Autónomo

La clase `controladorVehiculoAutonomo` que existía en la versión simple de C++ ha desaparecido. En su lugar existe un controlador con algunos de los métodos que tenía la clase controlador original, cambiando sus llamadas a los dispositivos del vehículo por peticiones de servicio de ROS, añadiendo los métodos necesarios de ROS para la comunicación entre el controlador y el vehículo y eliminado los métodos innecesarios. En la Figura 60 se puede ver un ejemplo de cómo cambian las llamadas a los dispositivos del vehículo en la versión de ROS.

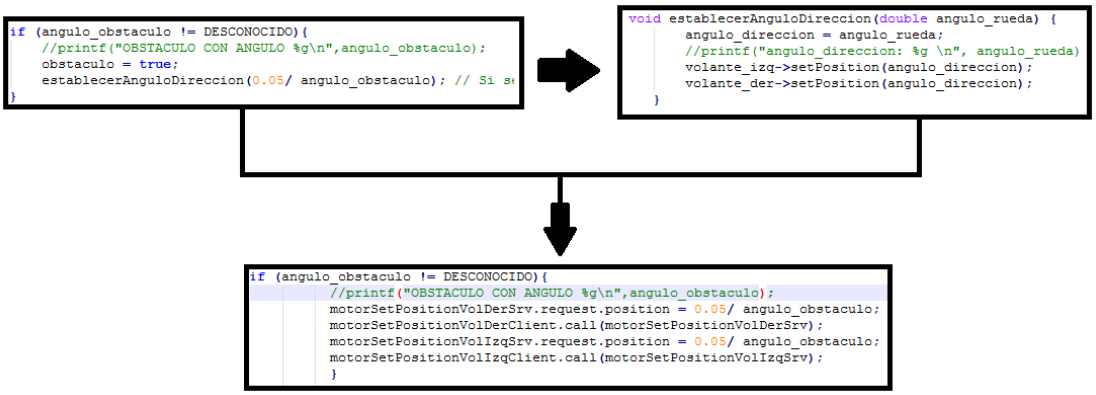


Figura 60: Cambios para adaptar el código a ROS

Como se puede observar en la Figura 60 para establecer el ángulo de las ruedas se ha pasado de utilizar una llamada a una función que le asignaba el valor directamente a cada dispositivo motor, a crear y enviar una solicitud al nodo de ROS correspondiente para que se modifique el valor del ángulo de las ruedas. De esta manera, la función ya no es necesaria (se ha hecho de esta manera en lugar de mediante una función porque para poder utilizar los mensajes, antes deben haberse creado y no existe la posibilidad de crearlos antes de implementar la función).

Salvo estos cambios en la asignación de los valores de los dispositivos del vehículo, la funcionalidad del controlador como tal es exactamente la misma. Se han eliminado los métodos `establecerAnguloDireccion` (por no ser necesario como se ha mencionado antes) y `ejecutar` (porque su funcionalidad ha pasado al método `main`). El resto de métodos no han sido modificados.

5.3.4.3. Función main

La función `main` sigue siendo la función principal del controlador, pero ha sufrido grandes cambios respecto a la implementación básica en C++. Al no existir ya la clase `controladorVehiculoAutonomo` para instanciar y llamar a su función `ejecutar`, ahora la función `main` contiene de forma explícita toda la funcionalidad de esa función `ejecutar`. Además de contener la funcionalidad de la antigua función `ejecutar`, también implementa todo lo necesario para activar los dispositivos del vehículo autónomo mediante ROS, así como la búsqueda del controlador del vehículo y la subscripción al hilo creado por este para poder establecer así la comunicación mediante el controlador del vehículo autónomo y el controlador de ROS que utiliza éste dentro de Webots.

5.3.4.4. Controlador ROS

Este controlador de ROS es un controlador que viene predefinido en el programa Webots. Puede ser usado tanto por un Robot (como es el caso del vehículo autónomo), un Supervisor o un DifferentialWheels (un robot que usa un sistema de ruedas diferenciales). Este controlador contiene todo lo necesario para conectarse al Nodo Maestro de ROS, darle un nombre al nodo correspondiente al controlador, crear un hilo de ROS y suscribir en él a dicho nodo, así como para crear los nodos de ROS que se corresponden con los dispositivos del vehículo autónomo.

Como se ha mencionado antes, este controlador esta predefinido en Webots y no es necesario modificarlo de ninguna manera para que funcione con el vehículo autónomo dentro de la simulación.

En la Figura 61 se puede ver de forma gráfica lo que ocurre mientras se está ejecutando el controlador de ROS, el controlador del vehículo autónomo y la simulación está activa.

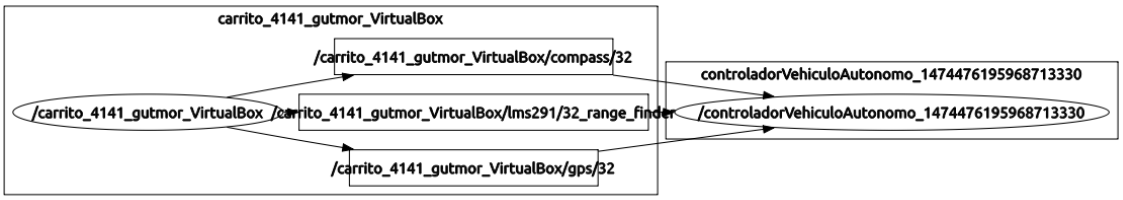


Figura 61: Esquema de comunicación de nodos en ROS

Como se puede observar, existen dos partes en las conexiones que se están llevando a cabo. Por una parte, tenemos el nodo llamado *carrito_4141_gutmor_VirtualBox*, que se corresponde con el nodo generado por el controlador ROS de Webots para la simulación que se está llevando a cabo en el programa. Ese nodo identifica al vehículo autónomo dentro de la simulación. Por otra parte, está el nodo *controladorVehiculoAutonomo* que se corresponde con el nodo del controlador implementado en C++ para el control del vehículo autónomo. Se pueden distinguir además tres hilos a través de los que se están comunicando los dos nodos antes mencionados. Estos hilos corresponden con los dispositivos brújula, laser SICK y GPS.

En la Figura 62 puede verse como se comunican ambos nodos mediante estos hilos. Por ejemplo, el nodo correspondiente al vehículo autónomo, en azul, publica los datos recogidos por el dispositivo brújula en su hilo correspondiente, en rojo, mientras que el nodo correspondiente al controlador, en verde, está suscrito a ese mismo hilo para poder utilizar los datos recogidos por la brújula para llevar a cabo su funcionalidad.

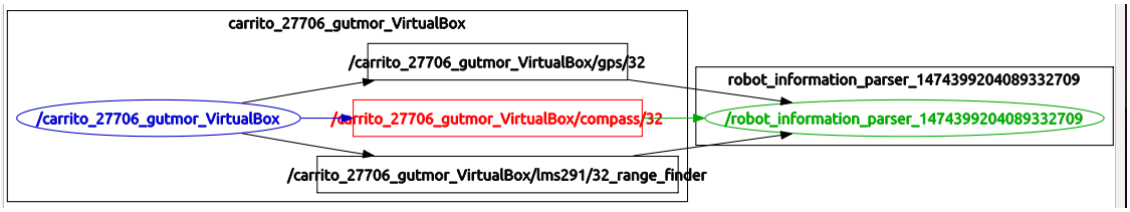


Figura 62: Publicación y subscripción a un hilo durante la ejecución del controlador en ROS

El resto de comunicaciones entre el nodo del controlador y el del carrito no aparecen en este gráfico que proporciona ROS porque se producen mediante peticiones de servicio y respuesta directa entre los nodos, tal y como se explica en la Figura 59.

Para explicar de una forma más sencilla como funciona el controlador mediante ROS, en la Figura 63 se presenta un esquema de lo que ocurre cuando se necesita cambiar el ángulo de las ruedas del vehículo.

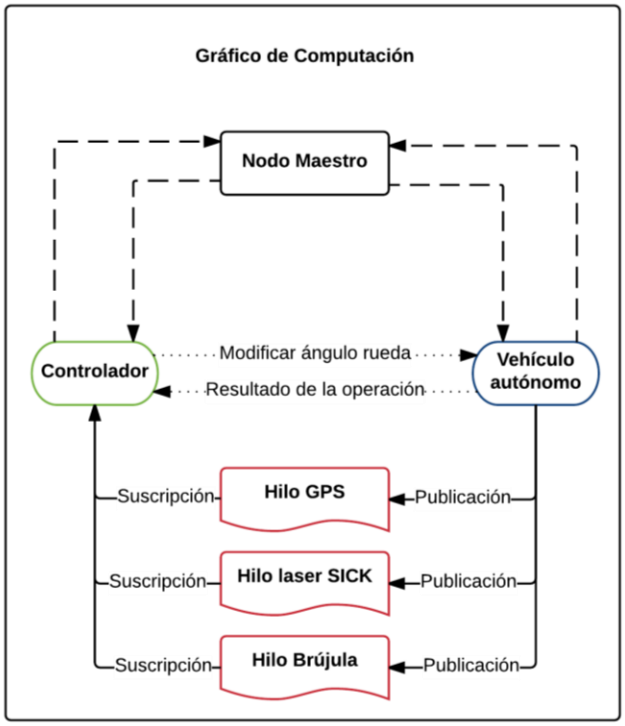


Figura 63: Ejemplo funcionamiento de ROS para modificar ángulo de rueda

El proceso que se lleva a cabo es el siguiente:

1. El nodo del controlador recibe los datos recogidos por los sensores del vehículo mediante su suscripción a los hilos correspondientes.
2. Tras analizar esos datos, se determina que se debe cambiar el ángulo de las ruedas del vehículo.
3. El nodo del controlador envía una petición al nodo del vehículo autónomo con un nuevo valor para el ángulo de las ruedas para que éste lo cambie en la simulación.
4. El nodo del vehículo autónomo envía una respuesta al nodo del controlador indicando si la operación requerida se ha llevado a cabo satisfactoriamente o no.

De este modo se realizan también el resto de operaciones sobre la simulación, como modificar su velocidad o avanzar instantes de tiempo.

5.4. Resultados y experimentación

En este apartado se muestran los resultados de diferentes ejecuciones del controlador implementado para el vehículo autónomo. Se detallará tanto la configuración del controlador como el proceso seguido por éste durante su ejecución.

5.4.1. Escenario 1

En esta ejecución del controlador se ha establecido como destino del vehículo autónomo el punto de ruta correspondiente al edificio Rey Pastor. La posición inicial del vehículo se ha establecido al otro lado del campus, junto al edificio Sabatini. La configuración inicial del experimento puede verse en la Figura 64.

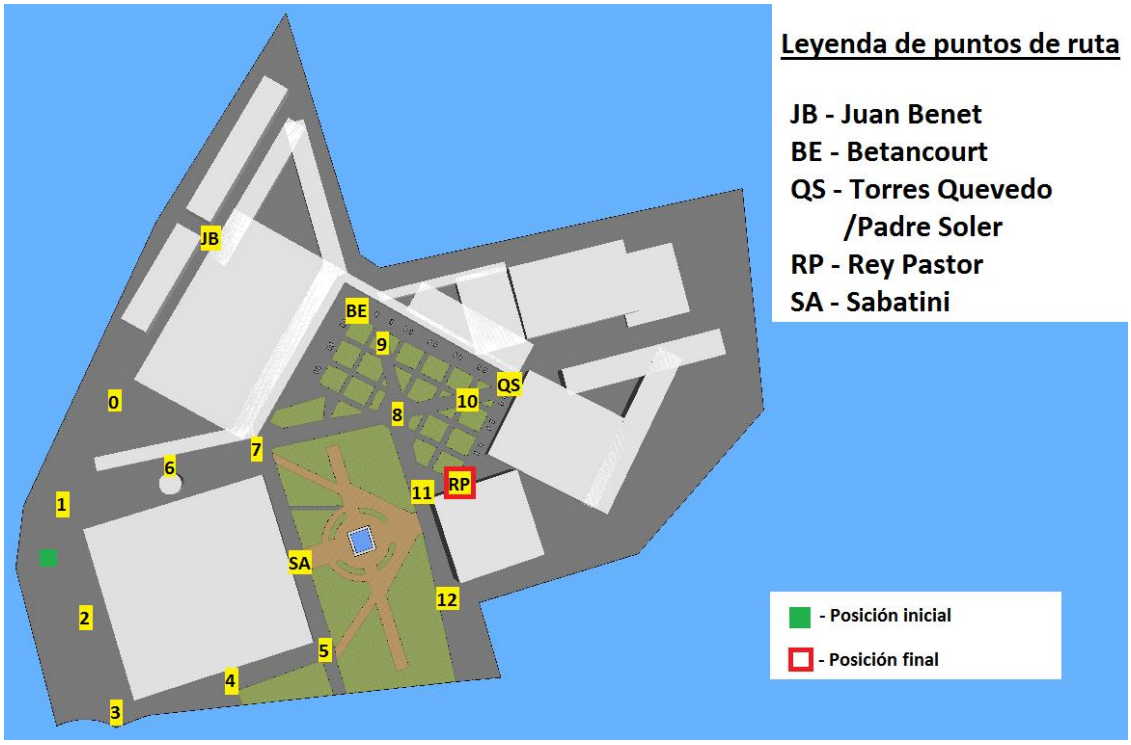


Figura 64: Configuración inicial del escenario de pruebas 1

Al ejecutar el controlador con esta configuración del escenario, el primer paso es buscar el punto de ruta inicial mediante la función **`puntoInicial.`** Esta función busca cual es el punto de ruta más cercano al que ocupa actualmente el vehículo y establece ese punto de ruta como el primer punto del recorrido. En este caso, el punto elegido es el punto de ruta número 1.

Una vez establecido el punto de ruta inicial, se ejecuta el método **`busquedaDeCamino`** hasta llegar al punto de ruta objetivo. Este método trabaja con una función heurística de la siguiente manera:

$$f(n) = g(n) + h(n)$$

Donde:

- **g(n):** es la distancia entre el punto de ruta en el que se encuentra el vehículo actualmente y el punto de ruta que se está evaluando como posible punto al que ir.
- **h(n):** es la distancia entre el punto de ruta que se está evaluando como posible punto al que ir y el punto de destino.

El algoritmo de búsqueda del punto de ruta a elegir como siguiente al que ir es el siguiente:

1. Se comprueba el valor heurístico de todos los puntos adyacentes al actual y se elige como destino aquel que tenga un valor heurístico menor.
2. En este punto existen dos posibilidades:
    a. Si existe un punto adyacente válido, se guarda el punto de ruta dentro de la lista de puntos de ruta a seguir.
    b. Si el punto en el que se encuentra ahora el algoritmo no tiene puntos de ruta adyacentes validos (porque sus puntos adyacentes han sido ya visitados o descartados), quiere decir que ese camino es incorrecto, ya que no es posible llegar al destino. Si se da este caso, este punto de ruta es eliminado del camino, se suma a una lista de descartados y se establece como punto actual el punto anterior del camino.

Este algoritmo se repite hasta que el punto de ruta actual coincide con el punto de ruta de destino. Cuando esto ocurre, comienza a ejecutarse el recorrido del vehículo por cada uno de los puntos de ruta definidos como camino.

En este escenario, la ruta elegida a seguir por el vehículo autónomo es la siguiente: **1-6-7-8-11-RP**. En la Figura 65 puede verse el proceso que ha seguido el algoritmo para calcular esta ruta.

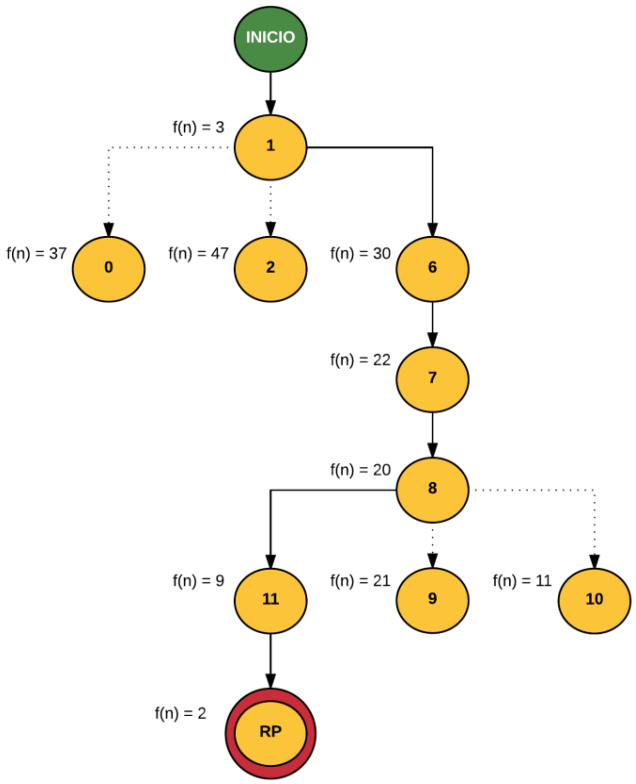


Figura 65: Búsqueda de camino del algoritmo en el escenario 1

El algoritmo de búsqueda de camino ha elegido una ruta correcta desde el punto inicial al destino. Tras ésta búsqueda de camino, comienza el movimiento del vehículo por el campus

siguiendo el camino marcado para llegar a su objetivo. La ruta recorrida dentro de la simulación puede verse en la Figura 66.

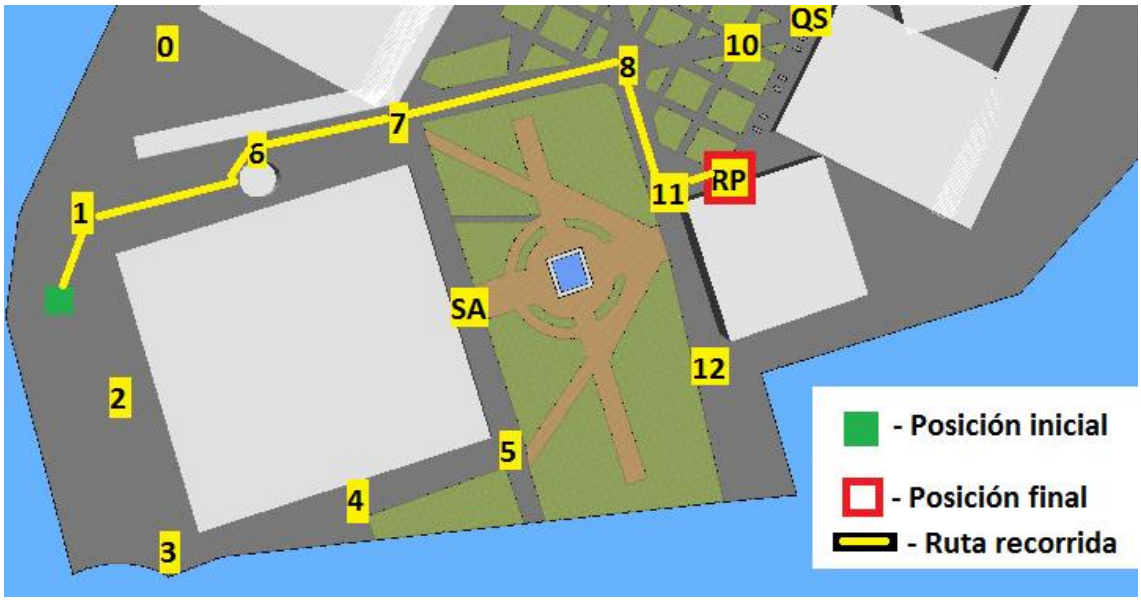


Figura 66: Recorrido del vehículo autónomo en el escenario 1

El vehículo autónomo recorre de forma correcta la ruta establecida por el algoritmo de búsqueda de caminos. En el recorrido desde el punto de ruta 1 al punto de ruta 6, el vehículo evita colisionar de forma con la Torre de la Caldera y se dirige a su objetivo. El resultado de le experimentación en este escenario es satisfactorio.

5.4.2. Escenario 2

En esta ejecución del controlador se ha establecido como destino del vehículo autónomo el punto de ruta correspondiente al edificio Juan Benet. La posición inicial del vehículo se ha establecido al otro lado del campus, junto al edificio Rey Pastor. Además, se ha añadido un obstáculo en el camino entre el punto de ruta 1 y el punto de ruta 0 que el vehículo tendrá que evitar La configuración inicial del experimento puede verse en la Figura 67.

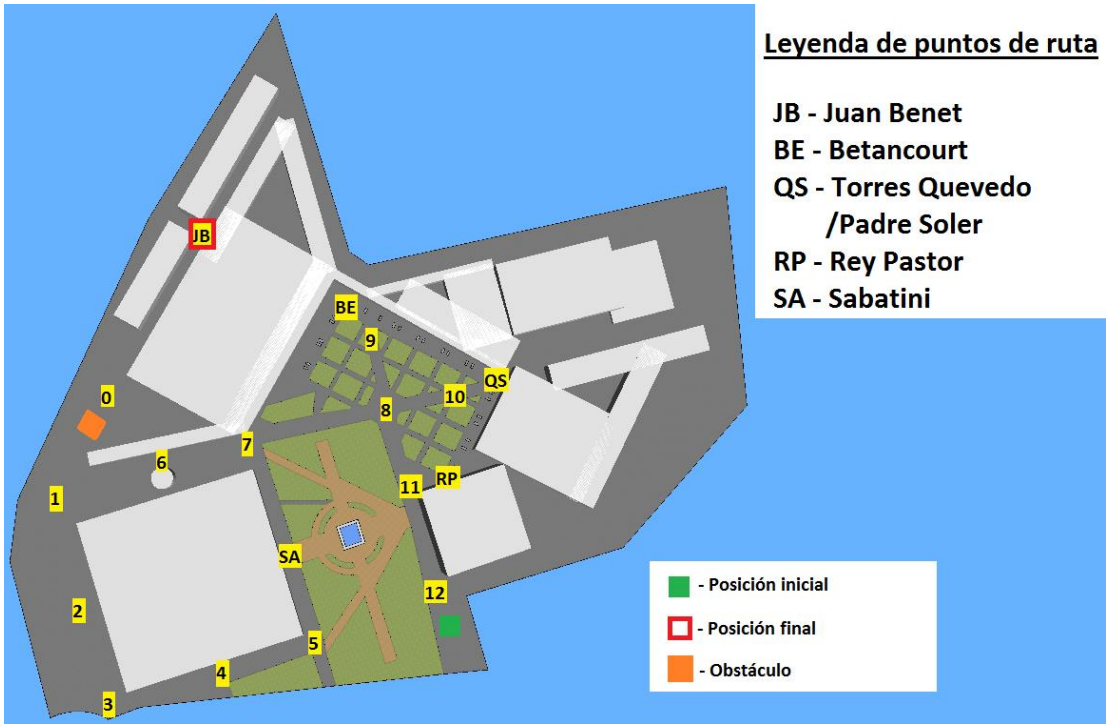


Figura 67: Configuración inicial del escenario de pruebas 2

En este escenario, la ruta elegida a seguir por el vehículo autónomo es la siguiente: **12-11-8-7-6-1-0-JB**. En la Figura 68 puede verse el proceso que ha seguido el algoritmo para calcular esta ruta.

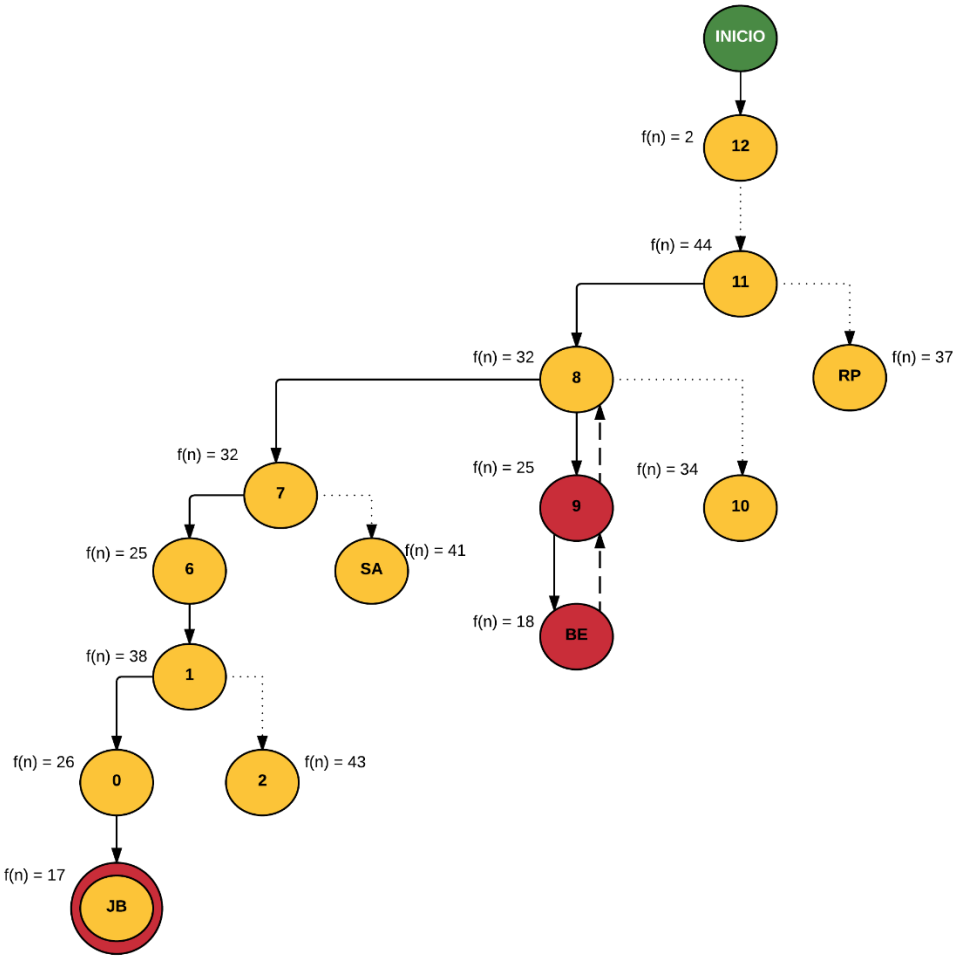


Figura 68: Búsqueda de camino del algoritmo en el escenario 2

Como se puede observar en la Figura 68, al llegar al punto de ruta 8, en primera instancia el algoritmo escoge como objetivo el punto de ruta 9 debido a que el valor de su función heurística es la menor de las opciones disponibles. El punto de ruta 9 solo tiene un punto adyacente, el correspondiente al edificio Betancourt, por lo que en la siguiente iteración ese es el punto seleccionado. Llegado a esta situación, el punto de ruta en el que se encuentra el algoritmo no tiene ningún otro adyacente y tampoco es el punto de destino, por tanto, elimina este punto de los posibles para elaborar una ruta y vuelve al anterior. Cuando vuelve al punto de ruta 9, este punto ya no tiene adyacentes al haberlos eliminado en el paso anterior, por lo que este punto es eliminado y el algoritmo vuelve al punto de ruta 8, desde donde continúa elaborando la ruta hasta el final.

El algoritmo de búsqueda de camino ha elegido una ruta correcta desde el punto inicial al destino. Tras ésta búsqueda de camino, comienza el movimiento del vehículo por el campus siguiendo el camino marcado para llegar a su objetivo. La ruta recorrida dentro de la simulación puede verse en la Figura 69.

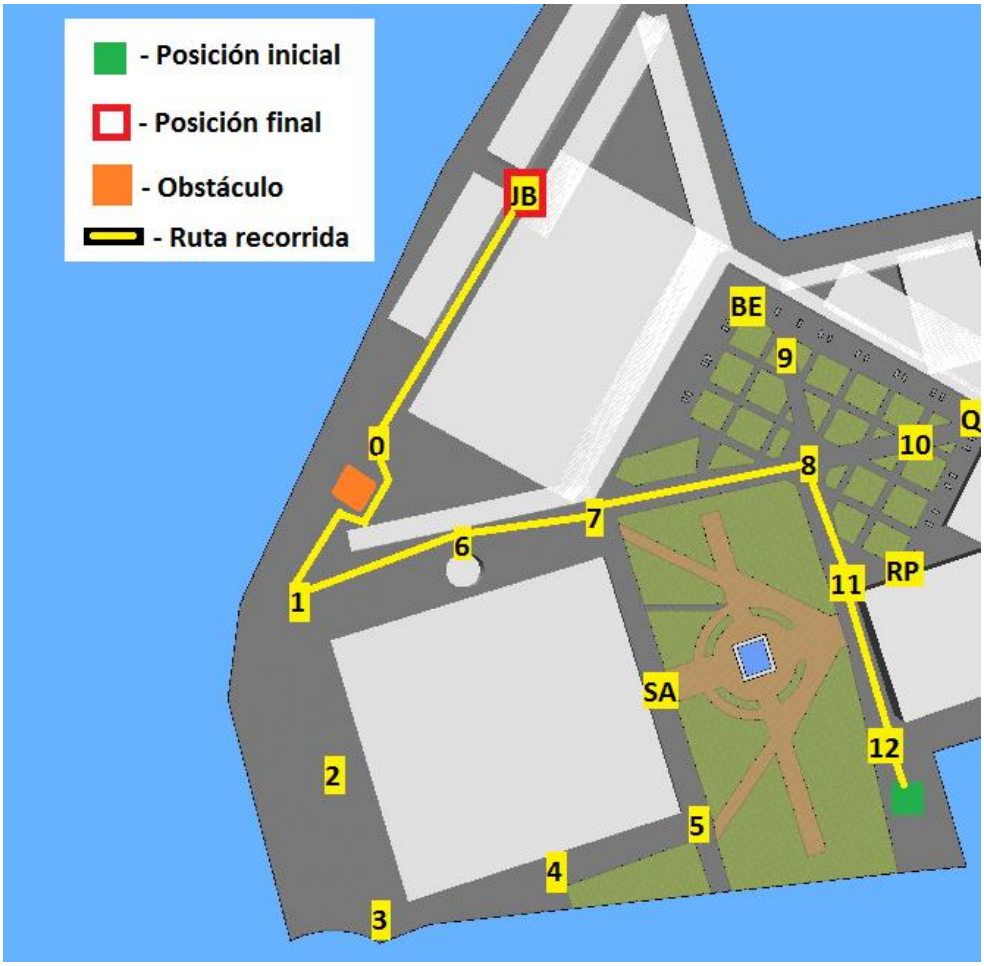


Figura 69: Recorrido del vehículo autónomo en el escenario 2

El vehículo autónomo recorre de forma correcta la ruta establecida por el algoritmo de búsqueda de caminos. En el recorrido dese el punto de ruta 1 al punto de ruta 0, el vehículo evita colisionar de forma correcta con el obstáculo introducido y se dirige a su objetivo. El resultado de le experimentación en este escenario es satisfactorio.

CAPÍTULO 6

Planificación y Presupuestos

6. Planificación y presupuesto

En esta sección de la memoria se detalla la planificación del trabajo realizado para el presente proyecto, así como el presupuesto realizado para la implementación de este proyecto completo, realizando una estimación de los costes de personal y equipo, tanto software como hardware.

6.1. Metodología de desarrollo de software

La metodología de desarrollo de software que se ha seguido para la realización del proyecto ha sido la tradicional metodología de desarrollo en cascada (ver Figura 70). La versión original de este modelo fue propuesta por Winston Royce en el año 1970, y posteriormente revisada por Barry Boehm en 1981 e Ian Sommerville en 1985 [58].

La metodología en cascada lleva a cabo un orden secuencial de las actividades, de forma que el inicio de una actividad coincide con el final de la actividad anterior. Al finalizar cada una de las etapas, se realizan una serie de revisiones para comprobar si se puede continuar o no con la siguiente. Se ha escogido esta metodología entre las diferentes existentes por su facilidad de seguimiento y entendimiento, además de por estar orientada a documentos por lo que es muy útil a la hora de redactar esta memoria sobre el proyecto.

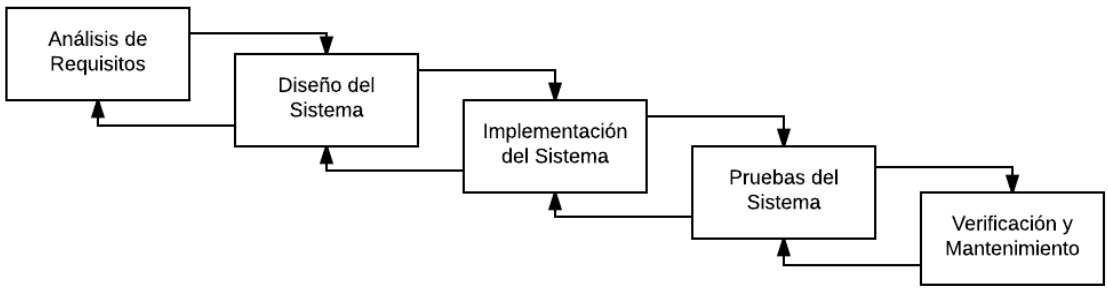


Figura 70: Esquema clásico de desarrollo en cascada

La metodología en cascada clásica se divide en las siguientes fases:

- **Análisis de requisitos.** En esta fase se estudian las necesidades del usuario final del software con el fin de determinar qué objetivos se deben alcanzar. Para ello, se deben organizar reuniones con el cliente, cuyas funciones en este caso las realiza el tutor del proyecto. Como resultado de llevar a cabo esta fase, se obtiene un documento de requisitos donde se detalla que trabajo debe llevarse a cabo y como debe llevarse a cabo.
- **Diseño del sistema.** En esta fase se divide el sistema en componentes o módulos, especificando cual es la función de cada uno de ellos. Como resultado de llevar a cabo esta fase se obtiene la especificación de los módulos que forman el sistema en base a los requisitos obtenidos en la fase anterior para su posterior implementación.

- **Implementación del sistema.** En esta fase se implementa el código fuente en base al diseño del sistema realizado en la fase anterior.
- **Pruebas del sistema.** En esta fase, tras implementar la solución diseñada, se realizan pruebas sobre dicha implementación con el fin de comprobar que cumple con los requisitos especificados en la primera fase de la metodología con el fin de encontrar y solucionar los posibles problemas en la funcionalidad de la implementación realizada.
- **Verificación y mantenimiento.** En esta fase el usuario final del software lleva a cabo una verificación del funcionamiento de éste. En caso de que el usuario final encuentre algún tipo de problema, el desarrollador llevará a cabo tareas de mantenimiento para subsanarlo.
- **Documentación.** Esta fase no aparece en la metodología clásica en cascada, pero es muy importante para el desarrollo de este proyecto que, una vez finalizado, se documente todo lo relativo a éste en forma de memoria.

6.2. Planificación del trabajo

El presente proyecto comenzó el día 15 de octubre de 2015 y finalizó el día 20 de septiembre de 2016. A continuación, en la Tabla 42, se muestra la duración total de cada una de las partes en las que se divide el proyecto, es decir, la parte correspondiente al análisis del sistema, la parte correspondiente al diseño de la simulación, la parte correspondiente a la implementación del controlador del vehículo autónomo y la parte correspondiente a la fase de documentación, indicando para cada una de ellas la fecha de inicio y finalización.

FASE	DURACIÓN (HORAS)	FECHA DE INICIO	FECHA DE FINALIZACIÓN
Análisis del sistema	14	15/10/2015	21/10/2015
Diseño de la simulación	171	22/10/2015	12/02/2016
Implementación del controlador	225	15/02/2016	27/05/2016
Pruebas del sistema	15	6/10/2016	10/06/2016
Documentación	60	13/06/2016	20/09/2016

Tabla 42: Planificación general del proyecto

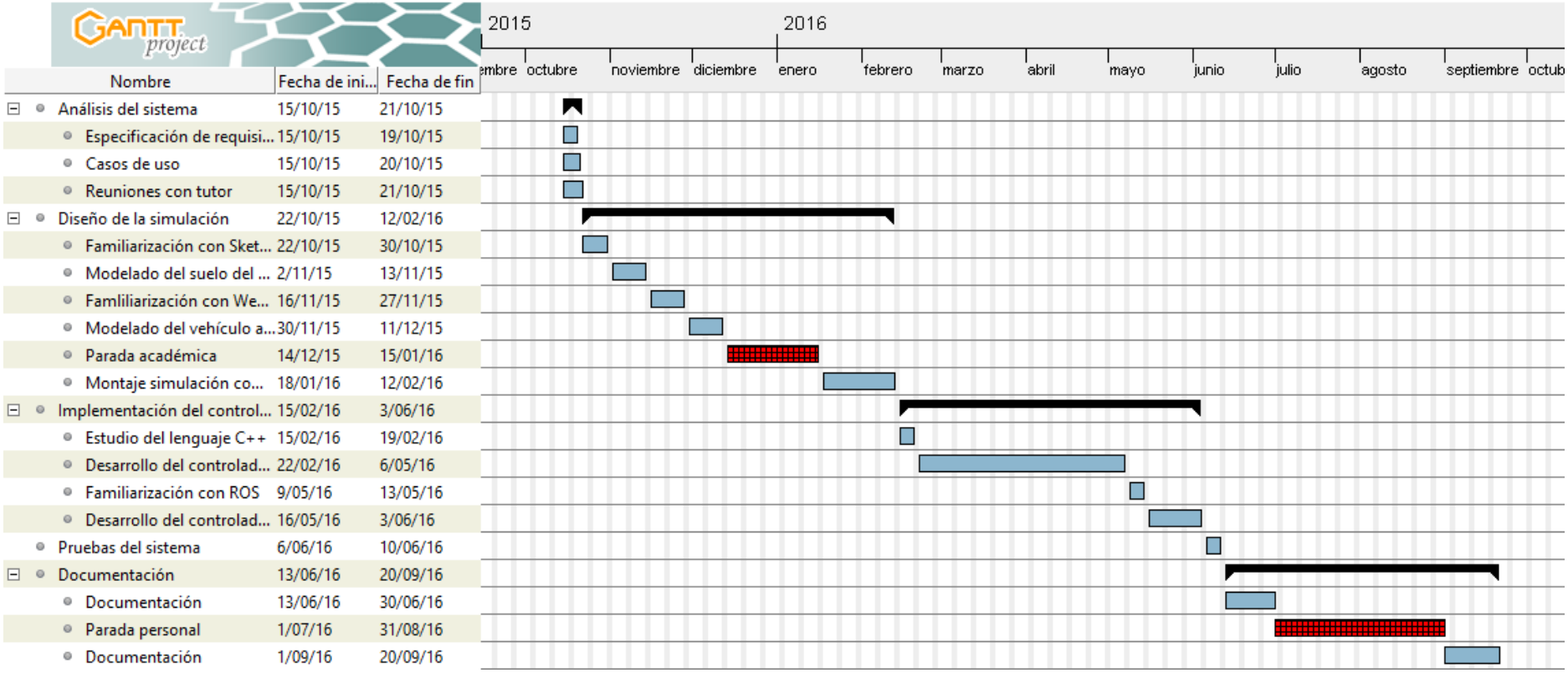
El total de horas de trabajo en el proyecto es de 480. Hay que destacar que se realizan una media de tres horas de trabajo diarias de lunes a viernes, mientras que no se trabaja en el proyecto los días sábado y domingo. Además, durante la fase de diseño de la simulación existe un periodo en el que no se trabaja por motivos académicos que comprende desde el día 14 de diciembre hasta el día 15 de enero. Entre la fase de implementación y la fase de documentación existe otro periodo donde no se trabaja por motivos personales que comprende desde el día 1 de julio hasta el día 1 de septiembre.

Una vez mostrada de forma general la duración de cada una de las fases que forman el proyecto, se va a proceder a detallar la planificación del proyecto en su conjunto. En la Tabla 43 se indica la duración en horas de cada una de las etapas que componen las fases del proyecto, así como su fecha de inicio y finalización.

TAREA	DURACIÓN (HORAS)	INICIO	FINALIZACIÓN
Análisis del sistema	**14**	**15/10/2015**	**21/10/2015**
Reuniones con tutor	5	15/10/2015	21/10/2015
Especificación de requisitos	6	15/10/2015	19/10/2015
Casos de uso	5	15/10/2015	20/10/2015
Diseño de la simulación	**171**	**22/10/2015**	**12/02/2016**
Familiarización con Sketchup	21	22/10/2015	30/10/2015
Modelado del suelo del campus	30	02/11/2015	13/11/2015
Familiarización con Webots	30	16/11/2015	27/11/2015
Modelado del vehículo autónomo	30	30/11/2015	11/12/2015
Montaje simulación completa	60	18/01/2016	12/02/2016
Implementación del controlador	**225**	**15/02/2016**	**03/06/2016**
Estudio del lenguaje C++	15	15/02/2016	19/02/2016
Desarrollo del controlador en C++	150	22/02/2016	06/05/2016
Familiarización con ROS	15	09/05/2016	13/05/2016
Desarrollo del controlador en ROS	45	16/05/2016	3/06/2016
Pruebas del sistema	**15**	**06/06/2016**	**10/06/2016**
Documentación	**60**	**13/06/2016**	**20/09/2016**

Tabla 43: Planificación detallada del proyecto

En la Figura 71 se muestra el diagrama de Gantt de la planificación descrita.



Nombre	Fecha de ini...	Fecha de fin
Análisis del sistema	15/10/15	21/10/15
Especificación de requisi...	15/10/15	19/10/15
Casos de uso	15/10/15	20/10/15
Reuniones con tutor	15/10/15	21/10/15
Diseño de la simulación	22/10/15	12/02/16
Familiarización con Sket...	22/10/15	30/10/15
Modelado del suelo del ...	2/11/15	13/11/15
Famliliarización con We...	16/11/15	27/11/15
Modelado del vehículo a...	30/11/15	11/12/15
Parada académica	14/12/15	15/01/16
Montaje simulación co...	18/01/16	12/02/16
Implementación del control...	15/02/16	3/06/16
Estudio del lenguaje C++	15/02/16	19/02/16
Desarrollo del controlad...	22/02/16	6/05/16
Familiarización con ROS	9/05/16	13/05/16
Desarrollo del controlad...	16/05/16	3/06/16
Pruebas del sistema	6/06/16	10/06/16
Documentación	13/06/16	20/09/16
Documentación	13/06/16	30/06/16
Parada personal	1/07/16	31/08/16
Documentación	1/09/16	20/09/16

Figura 71: Diagrama de Gantt de la planificación propuesta

6.3. Presupuesto

En este apartado de la memoria se presenta el presupuesto del proyecto. Para elaborar este presupuesto se calcularán los costes de personal, identificando los roles de las personas que participan en el proyecto y estableciendo sus sueldos, los costes de material y, finalmente, los costes totales.

6.3.1. Coste de personal

Dentro de la investigación universitaria los roles que se distinguen en el desarrollo de un proyecto son los siguientes:

- **Titular:** encargado de llevar a cabo las labores de seguimiento y evaluación del trabajo.
- **Ayudante:** encargado de llevar a cabo las labores de análisis, diseño, implementación y pruebas del proyecto.

A continuación, se puede ver en la Tabla 44 los salarios de cada uno de los roles anteriormente descritos. Estos salarios han sido calculados a partir de las tablas salariales publicadas en el Boletín Oficial del Estado número 36 del miércoles 11 de febrero de 2015, Sección III, páginas 11738 – 11750.

ROL	SALARIO BASE (€)	SALARIO/HORA (€)	HORAS TRABAJADAS	TOTAL (€)
Titular	1.186,57	7,42	17	126,14
Ayudante	703,07	4,46	480	2.139,21
COSTE TOTAL				**2.265,35**

Tabla 44: Cálculo de los costes de personal

El coste de personal asciende a un total de 2.265,35€.

6.3.2. Coste material

A continuación, se realiza un cálculo de los costes relativos a los materiales empleados para llevar a cabo el proyecto. Dentro de estos materiales se engloban tanto el equipo hardware utilizado para llevar a cabo el proyecto como el coste de los programas utilizados. De cada uno de estos materiales se tiene en cuenta su precio, su periodo de amortización y el tiempo que han sido utilizados para calcular su coste real dentro del proyecto siguiendo la siguiente formula:

$$coste\ material = \frac{precio\ material}{periodo\ de\ amortización} \times tiempo\ de\ uso$$

En la Tabla 45 se muestra el cálculo de los costes de material para este proyecto. Hay que tener en cuenta que el equipo hardware tiene un periodo de amortización de tres años mientras que los productos y licencias software tiene un periodo de amortización de seis años.

PRODUCTO	PRECIO (€)	PERIODO AMORTIZACIÓN (MESES)	TIEMPO DE USO (MESES)	COSTE MATERIAL (€)
PC de desarrollo	950	36	10	263,89
Televisor LED LG 22 pulgadas	135	36	10	37,45
Licencia Windows 10 PRO	279	72	10	38,75
Licencia Webots PRO	2.300	72	7	223,62
Licencia Office 2016 PRO	539	72	2	14,97
Licencia SketchUp PRO	657	72	2	18,25
COSTE TOTAL				**596,93**

Tabla 45: Cálculo de los costes de material

Los costes de material ascienden a un total de 596,93€.

6.3.3. Coste total del proyecto

Una vez calculados los costes tanto de personal como materiales, que dan lugar a los llamados costes directos, el coste total del proyecto es la suma de éstos.

CONCEPTO	COSTE TOTAL (€)
Costes de personal	2.265,35
Costes de material	596,93
COSTE TOTAL	**2.862,28**

Tabla 46: Cálculo del coste total del proyecto

El coste total del proyecto asciende a un total de 2.862,28€

CAPÍTULO 7

Conclusiones y Trabajos Futuros

7. Conclusiones y trabajos futuros

En este apartado se exponen las conclusiones extraídas de la realización de este trabajo de fin de grado, tanto las conclusiones técnicas como las conclusiones personales, así como las futuras líneas de trabajo que se pueden seguir a partir del proyecto realizado.

7.1. Conclusiones técnicas

Una vez finalizado el proyecto, se puede concluir que se han cumplido todos los objetivos propuestos al comienzo de éste de una forma satisfactoria.

El primer objetivo cumplido ha sido el modelado del entorno simulado del campus de la Escuela Politécnica Superior de la Universidad Carlos III de Madrid que se asemeja en gran medida al campus real. De igual manera se ha realizado de forma satisfactoria el modelado de un robot en forma de vehículo autónomo, totalmente funcional, que se asemeja a los vehículos autónomo reales propiedad de la Universidad Carlos III de Madrid.

Otro de los objetivos que se han llevado a cabo de forma satisfactoria es el de la implementación de un controlador para un vehículo autónomo capaz de circular por el entorno simulado del campus de la Escuela Politécnica Superior de la Universidad Carlos III de Madrid esquivando los obstáculos que se encuentra en el camino mientras se dirige a un punto concreto dentro de dicho campus.

El último objetivo propuesto y que también se ha conseguido de forma satisfactoria es adaptar el controlador del vehículo al marco de trabajo de ROS, permitiendo así que el futuro trabajo sobre el controlador y su aplicación a un vehículo real sea mucho más rápida y sencilla, y que la simulación realizada pueda ser traspasada al mundo real de una forma directa.

7.2. Conclusiones personales y conocimientos adquiridos

Desde un punto de vista personal, las conclusiones extraídas al finalizar este proyecto son muy positivas. Haber terminado este proyecto de forma tan satisfactoria supone un gran broche a los seis años que he pasado en esta universidad y la consecución, por fin, del grado en Ingeniería Informática.

Durante estos seis años nunca había tenido que enfrentarme a un proyecto de esta envergadura, sin más ayuda que la orientación de mi tutor y mis propias capacidades de investigación y aprendizaje. Esta experiencia me ha hecho ser partícipe de primera mano de lo que supone llevar a cabo la realización de un proyecto de investigación dentro de un grupo de trabajo como es el grupo CAOS de la Universidad Carlos III, y tomar experiencia para futuros proyectos, tanto en la universidad como en futuros puestos de trabajo.

Pero no solo he adquirido experiencia de trabajo, también he aprendido a manejar herramientas de las que, antes de este proyecto, no había oído hablar como son SketchUp, Webots o ROS. La experiencia adquirida con estas herramientas, así como con el lenguaje de programación C++, son un valor extra que me llevo de la realización de este proyecto.

7.3. Trabajos futuros

El trabajo realizado en este proyecto sienta las bases para distintas líneas de trabajo futuras. La principal línea de trabajo que se puede seguir a partir del trabajo desarrollado en este proyecto es la de implementar el controlador del vehículo autónomo en un vehículo autónomo físico ya que, como se han comentado anteriormente, el haber implementado el controlador en el marco de trabajo de ROS hace que el proceso de pasar el controlador del vehículo simulado al vehículo real sea más sencillo y directo.

Otras líneas de trabajo futuras que pueden desarrollarse a partir del trabajo realizado son trabajos de mejora de la simulación y trabajos de refinamiento del controlador del vehículo autónomo. Algunos ejemplos de estas líneas de trabajo son:

- **Remodelar los edificios del campus.** El grado de detalle con el que están modelados los edificios del campus es tan alto que provoca que la ejecución de la simulación sea demasiado lenta debido a la carga en memoria. Un posible trabajo futuro es rehacer los modelos con un nivel de detalle menor que permita una ejecución más fluida del controlador.
- **Añadir nuevas funcionalidades.** Es posible que, dependiendo de las necesidades de los trabajos futuros y de las nuevas funcionalidades requeridas, sea necesario ampliar el controlador para que el vehículo realice nuevas acciones. El código del controlador está ampliamente comentado para que sea sencillo añadir o eliminar funcionalidades al controlador.
- **Refinar el controlador.** Es posible también refinar o mejorar el funcionamiento del controlador mediante la modificación de sus funciones y parámetros. La estructura y comentarios del código permiten que su modificación sea sencilla y amigable.

Cabe destacar también que el controlador puede ser utilizado en cualquier entorno simulado, sin limitarse únicamente en el entorno del campus de la Escuela Politécnica de la Universidad Carlos III de Madrid, ya que por la forma en la que está, solo sería necesario introducir los puntos de ruta correspondientes al nuevo escenario que se va a utilizar. Debido a esto, otro de los posibles trabajos futuros sería el modelado de diferentes entornos para usar el controlador del vehículo autónomo en ellos. Además de esto, el controlador también se puede adaptar a otros vehículos, no solo al modelado para este proyecto, simplemente cambiando en el código del controlador los parámetros relativos al vehículo al que se le va a aplicar el controlador (el espacio entre sus ejes, el nombre de sus elementos en la simulación, etc.).

Teniendo en cuenta todo lo anterior, se puede concluir que el trabajo realizado en este proyecto puede ser utilizado en un futuro para realizar la simulación de cualquier vehículo (de estructura similar al utilizado en el proyecto) en cualquier entorno simulado, realizando pequeñas modificaciones a la implementación realizada, además de poder adaptarse de forma sencilla a un vehículo real gracias a su integración con ROS.

8. Referencias

[1] World Health Organization, «10 facts on global road safety,» 2015.

[2] «ROS,» [En línea]. Available: www.ros.org. [Último acceso: 2016].

[3] «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Autonomous_car. [Último acceso: 2016].

[4] SAE International, «SAE International's Levels of Driving Automation for On-Road Vehicle».

[5] «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Advanced_driver_assistance_systems. [Último acceso: 2016].

[6] «AAA Foundation for Traffic Safety,» [En línea]. Available: https://www.aaafoundation.org/forward-collision-warning-systems. [Último acceso: 2016].

[7] «ADAS ONE,» [En línea]. Available: www.adasone.com. [Último acceso: 2016].

[8] «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Lane_departure_warning_system. [Último acceso: 2016].

[9] «HONDA,» [En línea]. Available: www.tridenthonda.co.uk. [Último acceso: 2016].

[10] «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Autonomous_cruise_control_system. [Último acceso: 2016].

[11] «Ibericar,» [En línea]. Available: blog.ibericar.es. [Último acceso: 2016].

[12] «Revista DGT,» [En línea]. Available: http://revista.dgt.es/es/motor/tecnologia-seguridad/2016/0419-Tecnologia-n235-sist-frenado-autonomo.shtml#.V9u-T5iLSM8. [Último acceso: 2016].

[13] «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Automatic_parking. [Último acceso: 2016].

[14] «SAE International,» [En línea]. Available: www.sae.org.

[15] «Continental Automotive,» [En línea]. Available: http://www.continental-automotive.com/www/automotive_de_en/themes/passenger_cars/ov_automated_driving_en/traffic_jam_assist_en.html. [Último acceso: 2016].

[16] «Volkswagen,» [En línea]. Available: thinknew.volkswagen.com. [Último acceso: 2016].

[17] «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Robotics. [Último acceso: 2016].

[18] J. L. Poza Luján y J. L. Posadas Yagüe, Revisión de las Arquitecturas del Control Distribuido., 2009.

[19] «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Computer_architecture. [Último acceso: 2016].

[20] R. A. Brooks, A robust layered control architecture for a mobile robot. IEEE Journal of Robotics and Automation, 1986.

[21] V. Braitenberg, Vehicles: Experiments in Synthetic Psycology, 1984.

[22] N. J. Nilsson, Principles of Artificial Intelligence, 1980.

[23] E. Chown, Making predictions in an uncertain world: Environmental structure and cognitive maps, 1999.

[24] J. S. Albus, A Theory of Intelligent Machine Systems, 1991.

[25] R. C. Arkin, Integrating Behavioural, Perceptual and World Knowledge in Reactive Navigation, 1990.

[26] R. Alami, R. Chatila, S. Fleury, M. Herrb, F. Ingrand, M. Khatib, B. Morisset, P. Montarlier y T. Simeon, «Around the lab in 40 labs... In IEEE International Conference on Robotics and Automation,» de *IEEE International Conference on Robotics and Automation*, San Francisco, 2000.

[27] R. P. Bonasso, J. Firby, E. Gat, D. Kortenkamp, D. Miller y M. Slack, «A Proven Three-tiered Architecture for Programming Autonomous Robots,» *Journal of Experimental and Theoretical Artificial Intelligence,* vol. 9, nº 2, 1997.

[28] K. Konolige y K. Myers, The Saphira Architecture for Autonomous Mobile Robots, 1998.

[29] H. Hexmoor, J. Lammens y S. C. Shapiro, Embodiment in GLAIR: A grounded layered architecture with integrated reasoning for autonomous agents, 1993.

[30] J. M. Lammens, H. H. Hexmoor y S. C. Shapiro, Of elephants and men, 1993.

[31] C. Laugier, T. H. Fraichard, P. H. Garnier, I. E. Paromtchik y A. Scheuer, Sensor-based control architecture for a car-like vehicle, 1999.

[32] M. Lindström, A. Orebäck y H. Christensen, «Berra: A research architecture for service robots,» de *International Conference on Robotics and Automation*, 2000.

[33] J. Connel, «SSS: A Hybrid Architecture Applied to Robot Navigation,» de *IEEE International Conference on Robotics and Automation*, 1992.

[34] D. W. Payton, «An Architecture for Reflexive Autonomous Vehicle,» de *IEEE International Conference on Robotics and Automation*, 1986.

[35] «Real Academia Española de la Lengua,» [En línea]. Available: http://dle.rae.es/?id=Xw14yph.

[36] «STISIM Drive,» [En línea]. Available: http://www.stisimdrive.com/products/stisim-drive-software. [Último acceso: 2016].

[37] «Systemstech,» [En línea]. Available: systemstech.com. [Último acceso: 2016].

[38] «National Advanced Driving Simulator,» [En línea]. Available: https://www.nads-sc.uiowa.edu/overview.php. [Último acceso: 2016].

[39] «Drive-sim,» [En línea]. Available: http://drivesimsimulator.com/. [Último acceso: 2016].

[40] «City Car Driving,» [En línea]. Available: http://citycardriving.com/products/citycardriving. [Último acceso: 2016].

[41] «Assetto Corsa,» [En línea]. Available: http://www.assettocorsa.net/en/. [Último acceso: 2016].

[42] «Project Cars,» [En línea]. Available: http://www.projectcarsgame.com/info.html. [Último acceso: 2016].

[43] «Microsoft,» [En línea]. Available: https://msdn.microsoft.com/en-us/library/bb483024.aspx. [Último acceso: 2016].

[44] «Roboworks,» [En línea]. Available: http://www.newtonium.com/public_html/Products/RoboWorks/RoboWorks.htm. [Último acceso: 2016].

[45] «Webots,» [En línea]. Available: https://www.cyberbotics.com/webots.php. [Último acceso: 2016].

[46] Dirección General de Tráfico, «Intrucción 15/V-113- - Autorización de pruebas o ensayos de investigación realizados con vehículos de conducción automatizada en vías abiertas al tráfico en general,» 2016.

[47] Gobierno de España, «BOE, Real Decreto 2822/1998,» 1998.

[48] «SketchUp,» [En línea]. Available: https://www.sketchup.com/es. [Último acceso: 2016].

[49] «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/VRML. [Último acceso: 2016].

[50] «Universidad Carlos III de Madrid,» [En línea]. Available: www.uc3m.es.

[51] V. Romero Pérez, *Creación de un entorno 3D para la simulación de tráfico urbano,* 2009.

[52] «Ez-Go,» [En línea]. Available: http://www.ezgo.com/. [Último acceso: 2016].

[53] «Bumblebee2 Firewire stereo vision camera systema,» [En línea]. Available: https://www.ptgrey.com/bumblebee2-firewire-stereo-vision-camera-systems. [Último acceso: 2016].

[54] P. Grey, *Bumblebee: stereo visión camera systems. Technical Description,* 2012.

[55] «SICK España,» [En línea]. Available: https://www.sick.com/es/es/. [Último acceso: 2016].

[56] Intelligence, S S;, *LMS 200/211/221/291 laser measurement systems. Technical Description,* 2006.

[57] «ROS.org,» [En línea]. Available: http://www.ros.org/. [Último acceso: 2016].

[58] Z. Cataldi, F. Lage, R. Pessacq y R. García Martínez, *Ingeniería de software educativo.*

Anexo A. Manual de Usuario

Este manual está orientado a cualquier usuario que quiera utilizar o modificar el controlador para el vehículo autónomo desarrollado en este proyecto. En él se explica de forma detalla los pasos que se deben seguir para hacer funcionar el controlador, tanto con ROS como sin él, así como los valores que hay que modificar si se desea utilizar un entorno diferente al del campus de la Escuela Politécnica de la Universidad Carlos III o un vehículo autónomo diferente al modelado.

A.1. Requisitos previos

Antes de poder utilizar el controlador desarrollado o el entorno simulado, es necesario preparar el equipo donde se va a llevar a cabo el trabajo. Para ello deben llevarse a cabo los siguientes pasos:

- **Instalación de Webots.** Para poder trabajar con el controlador y el entorno simulado, es necesario instalar la versión 7.4.3. de Webots (no se asegura el correcto funcionamiento del proyecto en versiones anteriores o posteriores). El manual de instalación del programa puede encontrarse en el siguiente enlace: https://www.cyberbotics.com/guide/installing-webots.php
- **Adquirir una licencia de Webots.** Una vez instalado Webots, es necesario adquirir una licencia para poder trabajar en el proyecto. Puede usarse también la prueba gratuita de 30 días que ofrece el programa si el trabajo no va a requerir más tiempo.
- **Instalación y configuración del entorno de ROS.** Si se va a utilizar la versión integrada con ROS, es necesario instalar la versión Jade de este marco de trabajo (no se asegura el correcto funcionamiento del proyecto en versiones anteriores o posteriores). Este marco de trabajo solo se encuentra disponible para los sistemas operativos Ubuntu y Debian. Una vez instalado, es necesario realizar la configuración de su entorno. Todos los pasos necesarios para la instalación y configuración del entorno de ROS vienen descritos en el manual de instalación puede encontrarse en el siguiente enlace: http://wiki.ros.org/ROS/Installation

Una vez realizados estos preparativos, se considera que el usuario ya está listo para poder ejecutar y trabajar con la simulación.

A.2. Ejecutar la simulación

En esta sección del manual se considera que el usuario ya tiene instalado y configurado correctamente Webots y el entorno de ROS. Si no es así, por favor vuelva al punto **A.1. Requisitos previos** de este mismo manual y siga los pasos que allí se le indican.

A.2.1. Ejecución sin ROS

Los pasos que debe seguir el usuario para poder trabajar con el proyecto del vehículo autónomo en un entorno simulado sin utilizar ROS son los siguientes:

1. **Abrir Webots y cargar la simulación.** El primer paso es abrir la versión de Webots instalada y licenciada mediante un doble clic en el icono correspondiente al programa dentro del ordenador del usuario. Una vez abierto y en la interfaz principal de Webots el usuario deberá seleccionar la opción ***File → Open World*** y seleccionar el archivo que contiene la simulación.

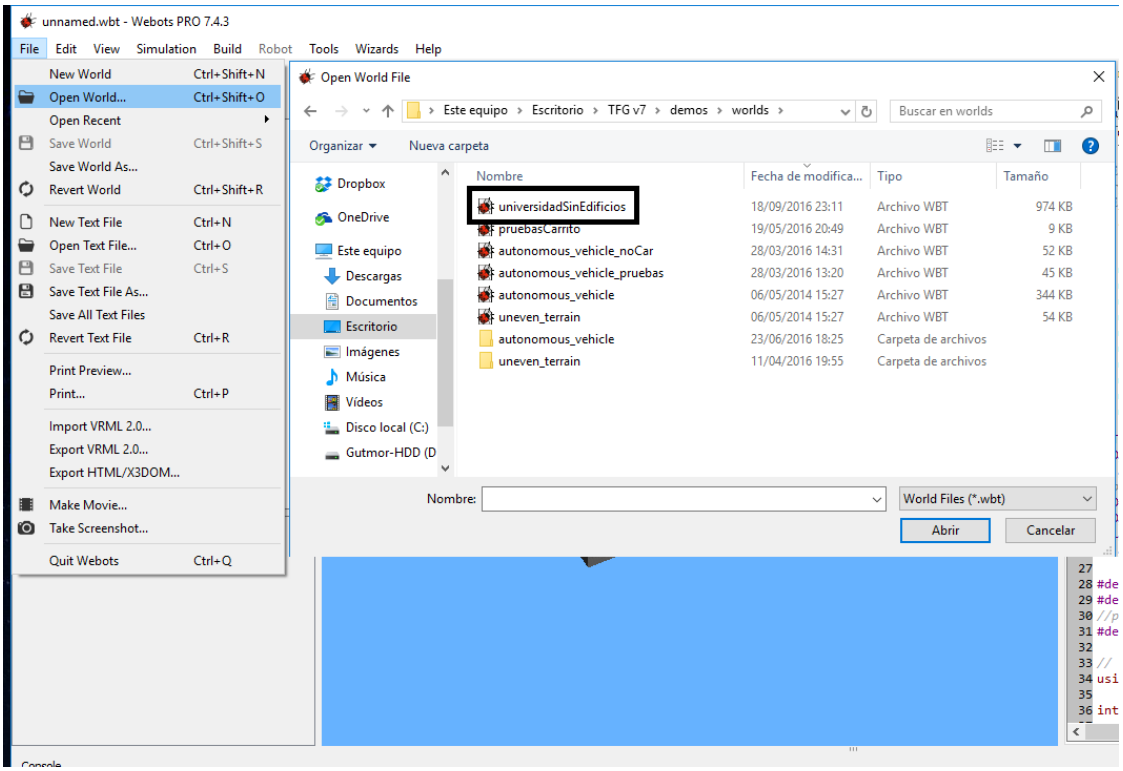


Figura 72: Proceso de carga de la simulación en Webots

2. **Crear el controlador.** Una vez se tiene el entorno simulado cargado en Webots, es necesario crear un nuevo controlador para poder ejecutarlo sobre el vehículo autónomo dentro de la simulación.
   2.1. El primer paso para crear un nuevo controlador es, en la pantalla principal de Webots, usar la opción ***Wizards → New Robot Controller.*** Una vez abierto el asistente de creación de controladores, se selecciona como lenguaje para el controlador **C++** y se le pone como nombre **controladorVehiculoAutonomo.**

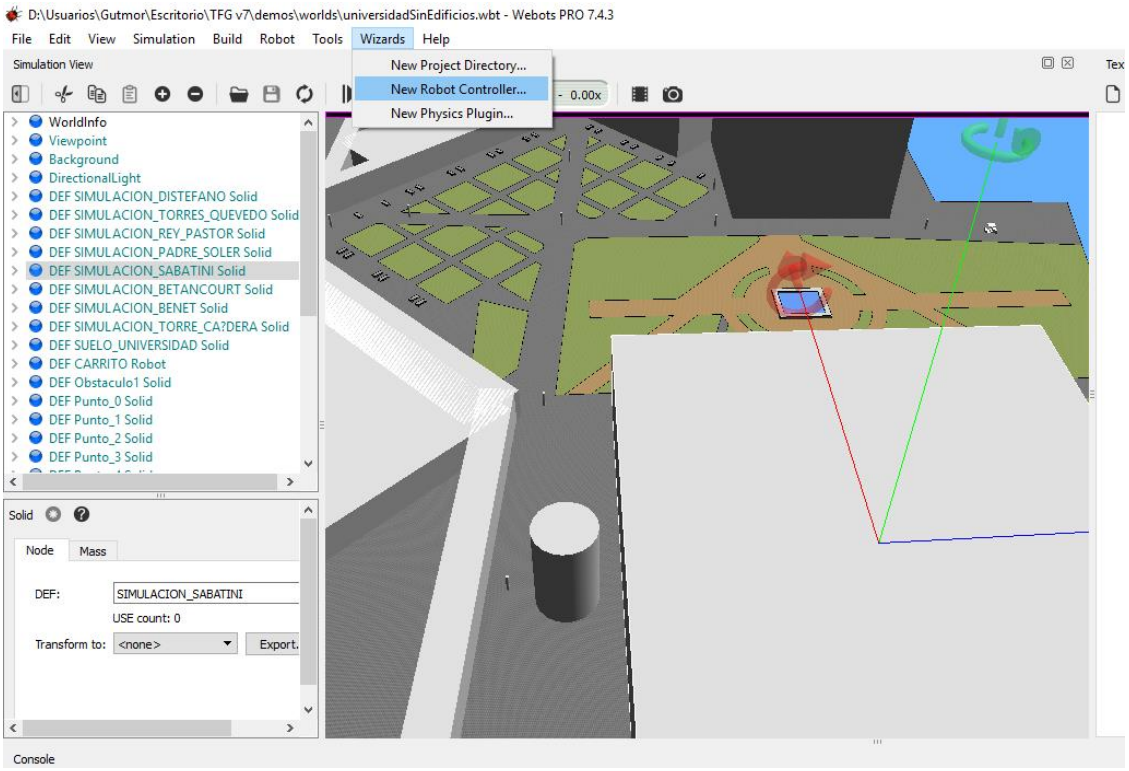


Figura 73: Creación de un nuevo controlador en Webots

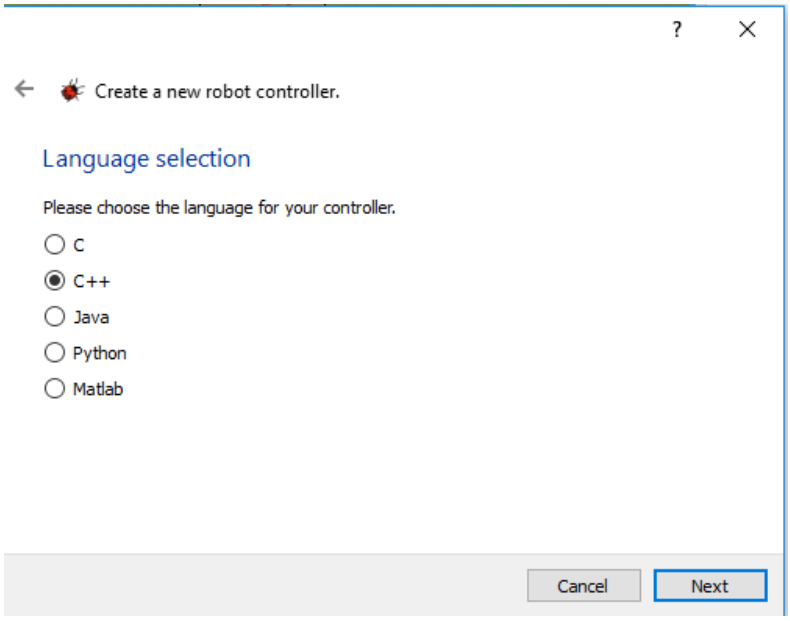


Figura 74: Selección del lenguaje del controlador

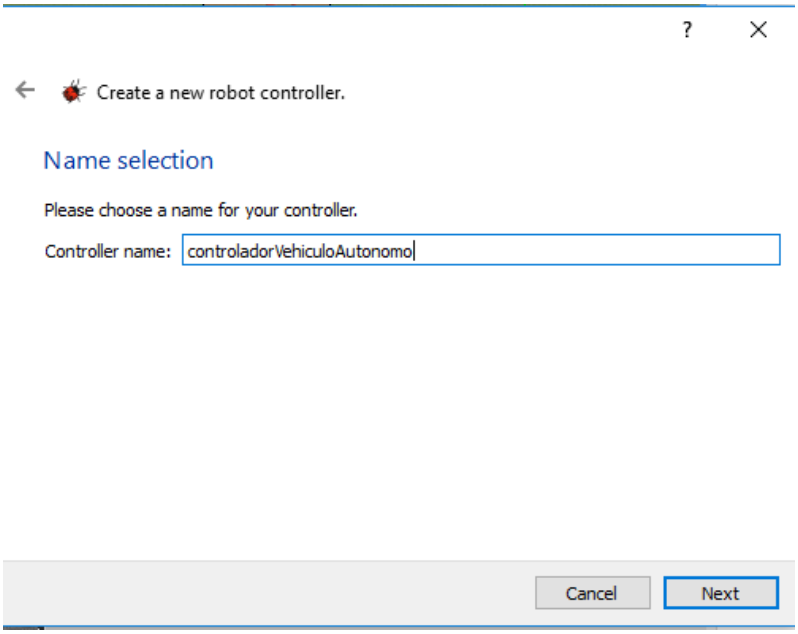


Figura 75: Elección de nombre para el controlador que se va a utilizar

2.2. Una vez realizados los pasos anteriores y finalizando la creación del controlador, se mostrará en la parte derecha de la interfaz principal de Webots el editor de texto con el código del controlador generado (si se ha seleccionado la opción ***Open 'controladorVehiculoAutonomo.cpp' in Text Editor***). Una vez en el editor de texto, lo que hay que hacer es borrar todo el contenido del fichero generado y sustituirlo por el código del fichero **controladorVehiculoAutonomo.cpp** proporcionado para este proyecto. Una vez copiado el contenido en el editor de texto, se debe pulsar en el icono de un disquete (opción ***Save***) para guardar los cambios y finalmente, **y muy importante**, en el icono de un engranaje (opción ***Build***) para compilar el proyecto. Al final de dicha compilación se le preguntará al usuario si quiere revertir la simulación, a lo que se indicará que sí.

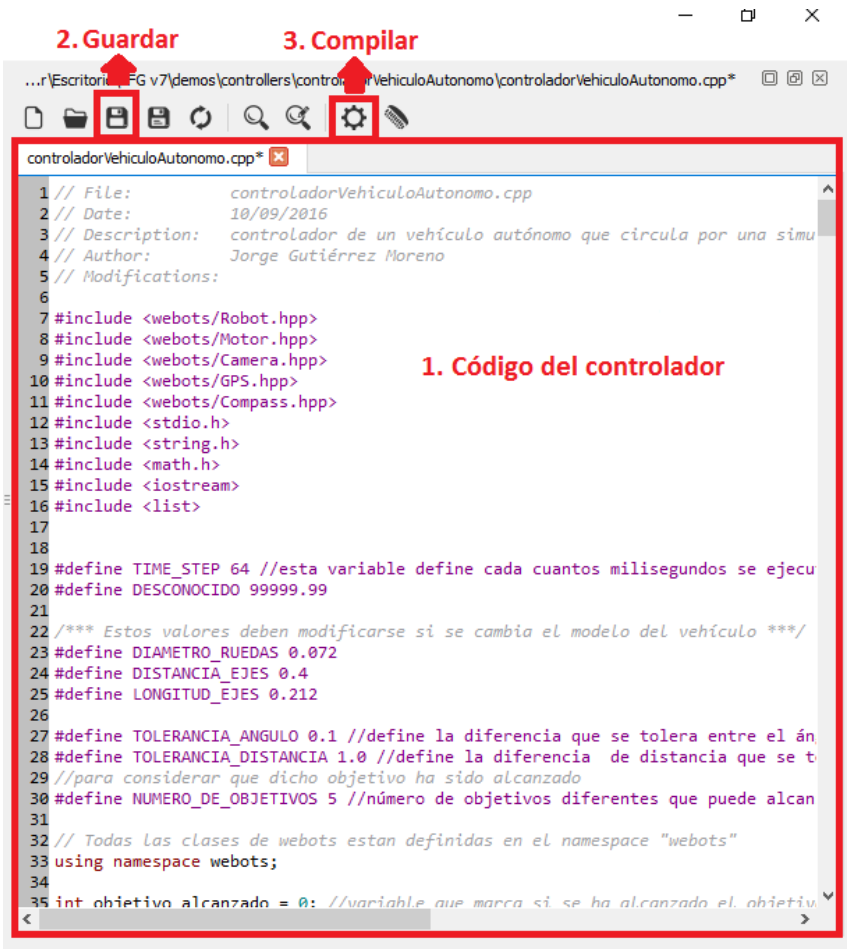


Figura 76: Creación y compilación del controlador para el vehículo autónomo

3. **Asignar el controlador.** Tras la creación y compilación del controlador, ahí que asignárselo al vehículo autónomo dentro de la simulación para que este lo use para circular por el entorno simulado. Para realizar la asignación, el usuario debe buscar el nodo correspondiente al vehículo autónomo dentro del árbol que define la simulación, donde se encuentra con el nombre de **DEF CARRITO ROBOT.** Dentro de sus propiedades el usuario tiene que seleccionar la propiedad ***controller*** y usar la opción ***Select...*** para seleccionar el controlador creado en el paso anterior. Una vez asignado el controlador, se guardan los cambios mediante el icono de un disquete (opción ***Save***).

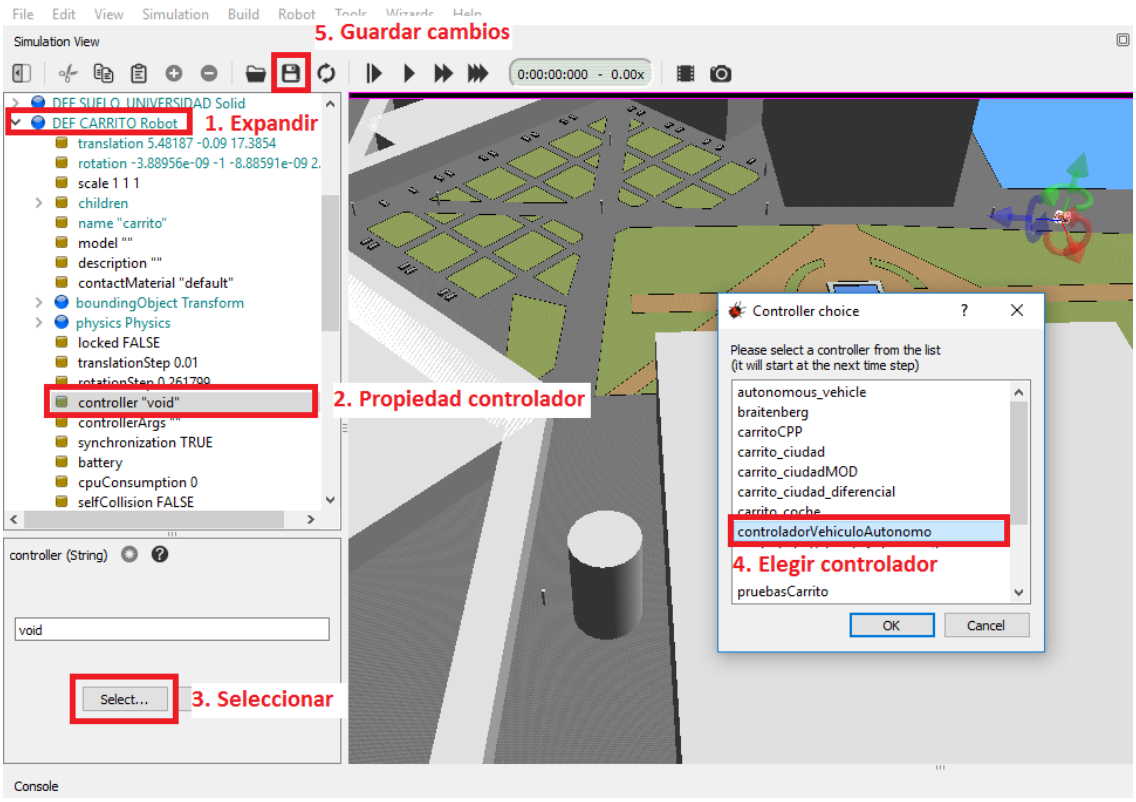


Figura 77: Asignación del controlador al vehículo autónomo dentro de la simulación

4. **Ejecutar la simulación.** Tras seguir los pasos anteriores, la simulación está lista para ser ejecutada. Para ello, el usuario debe pulsar en el botón correspondiente a la opción ***Run the simulation in real time***. Además de esta opción, el usuario también dispone de opciones para detener la simulación o aumentar la su velocidad. Una vez finalizada la simulación es **altamente recomendable** que el usuario la detenga mediante la opción ***Pause*** y devuelva la simulación a su estado original mediante la opción ***Reload***.

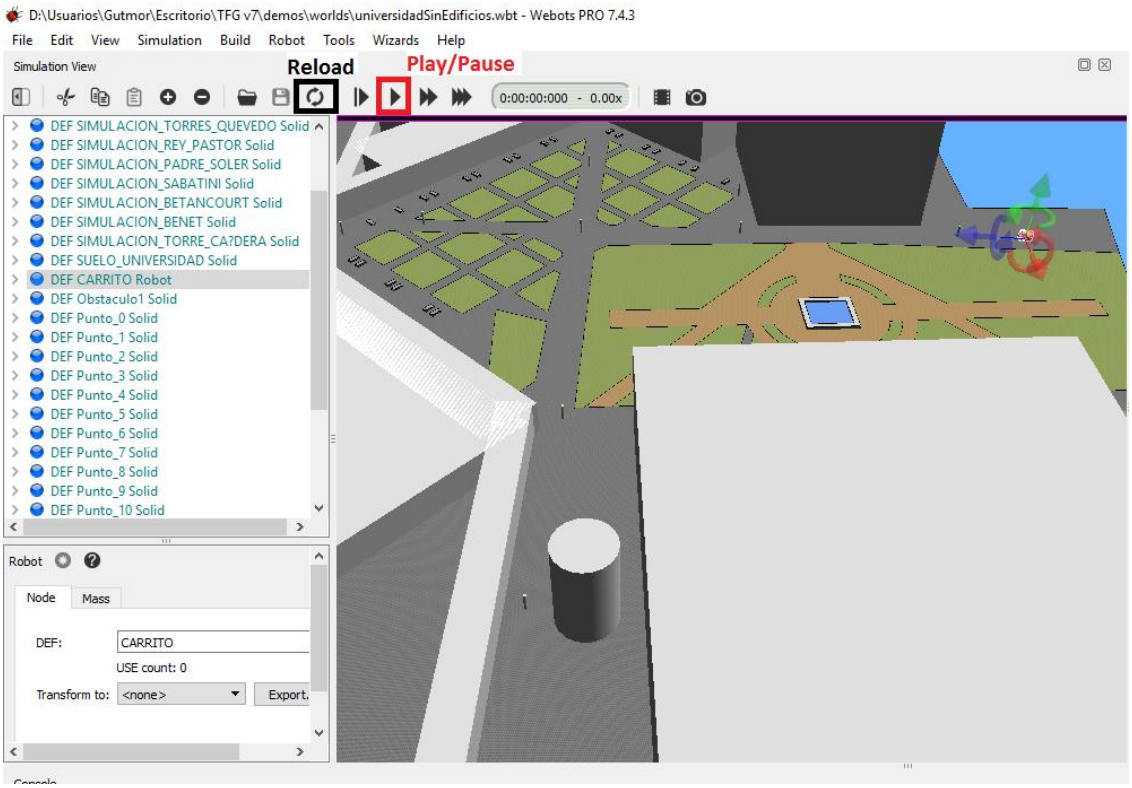


Figura 78: Ejecución del controlador del vehículo autónomo sin ROS

A.2.2. Ejecución con ROS

Todos los pasos expuestos en esta parte del manual han sido realizados en el sistema operativo Ubuntu LTS 14.04 LTS, pero son los mismos independientemente de la versión de Ubuntu o Debian que se utilice. Los pasos que debe seguir el usuario para poder trabajar con el proyecto del vehículo autónomo en un entorno simulado sin utilizar ROS son los siguientes:

1. **Iniciar el Nodo Maestro de Ros.** Es esencial para el funcionamiento de ROS que se lance su Nodo Maestro. Para ello, el usuario debe abrir una nueva terminal del sistema y escribir el comando ***roscore***. Automáticamente se lanzará el Nodo Maestro de ROS. Es de **vital importancia no cerrar esta terminal** ni detener su ejecución durante el tiempo que se quiera trabajar usando ROS, ya que este proceso es el que mantiene el sistema de ROS en funcionamiento.

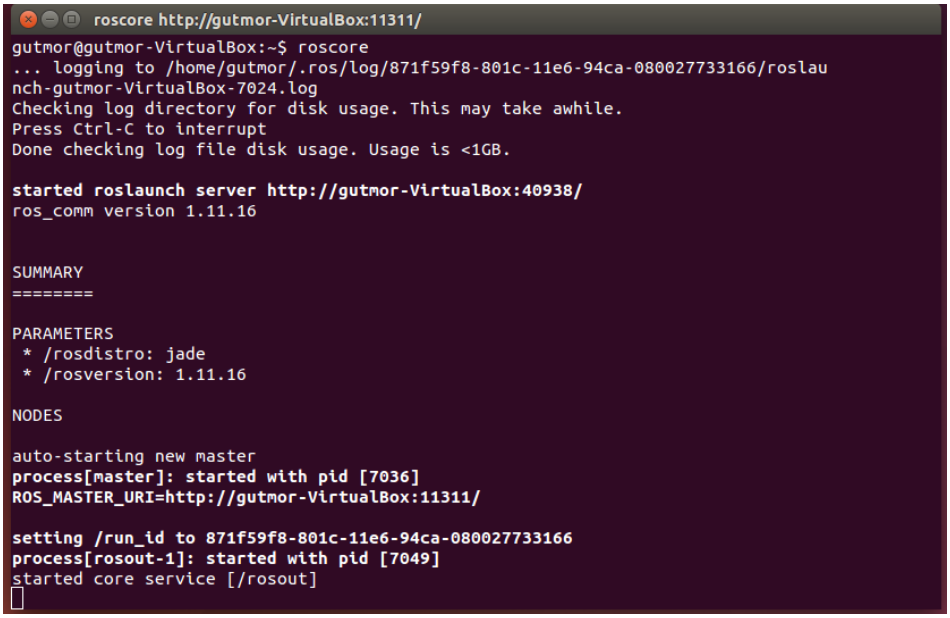


Figura 79: Ejecución del Nodo Maestro de ROS

2. **Ejecutar el controlador del vehículo autónomo.** Para que comience la ejecución de la simulación dentro de Webots, es necesario poner en marcha el nodo del controlador del vehículo autónomo mediante una terminal del sistema. Para ello el usuario debe irse a la carpeta de ROS dentro de su ordenador, y realizar la ejecución del nodo. Antes el usuario ha de asegurarse de que el nodo está compilado (mediante la instrucción ***catkin_make***) y de que la fuente está bien establecida (***source devel/setup.bash***). Para ejecutar el nodo, el usuario debe usar la instrucción ***rosrun nombre_paquete controladorVehiculoAutonomo***, donde ***"nombre_paquete"*** es el nombre del paquete raíz donde se encuentra el controlador.

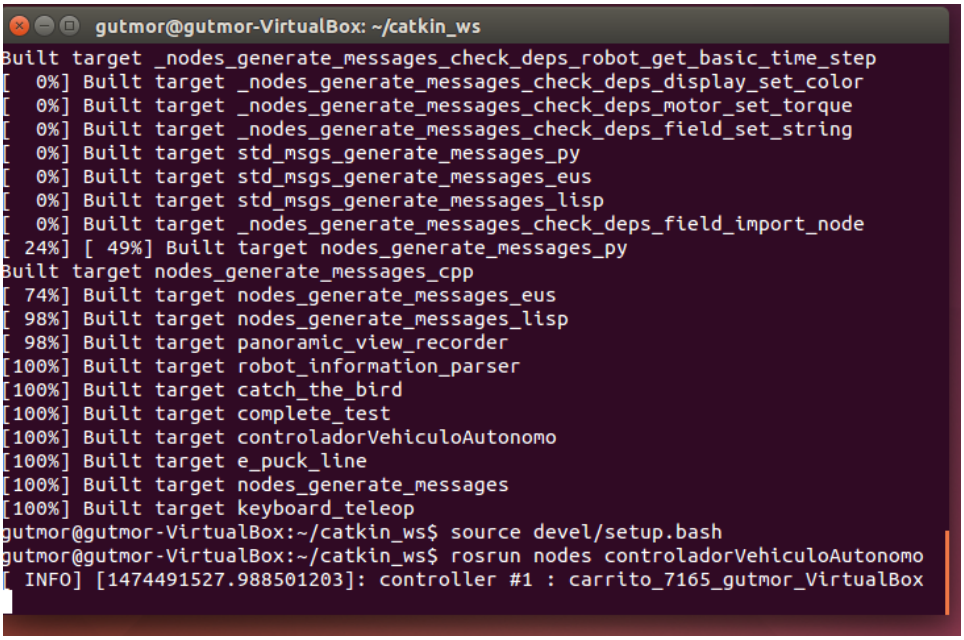


Figura 80: Ejecución del nodo del controlador mediante ROS

3. **Iniciar Webots y cargar la simulación.** Para iniciar Webots, el usuario debe abrir una nueva terminal del sistema y entrar en el directorio donde tiene instalado Webots. Una vez dentro del directorio, se debe ejecutar Webots como **súper usuario** mediante el comando ***sudo ./webots.*** Automáticamente se lanzará el programa Webots. Es de **vital importancia no cerrar esta terminal** ni detener su ejecución durante el tiempo que se quiera trabajar usando Webots, ya que este proceso es el que mantiene el programa en funcionamiento. Una vez abierto el programa, el proceso de carga de la simulación es el mismo que el que se ha explicado para la ejecución sin ROS de este mismo manual.

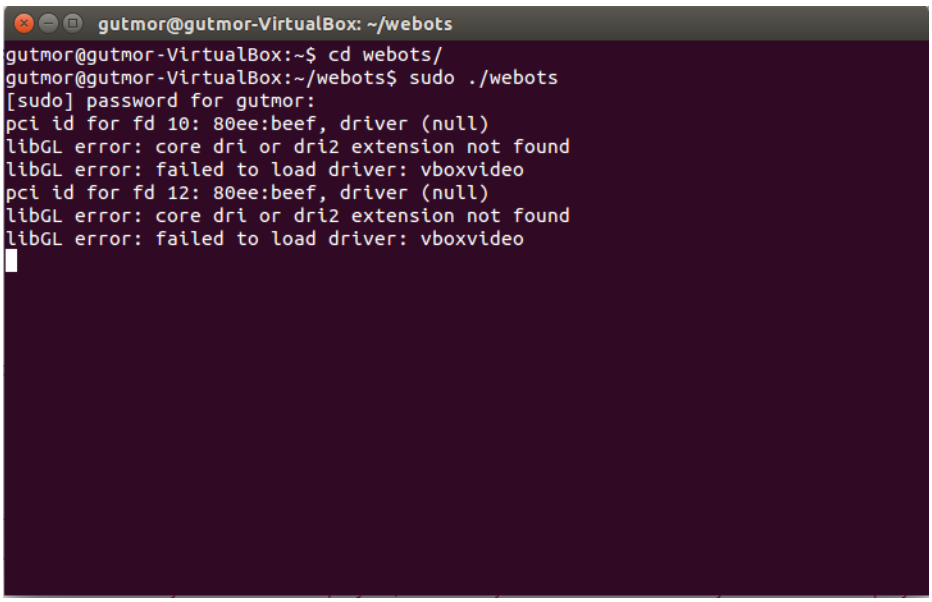


Figura 81: Ejecución de Webots mediante terminal

4. **Asignar el controlador.** Una vez cargada la simulación, el siguiente paso es asignarle el controlador de ROS al vehículo autónomo dentro de Webots. El proceso para la asignación del controlador es el mismo que el que se ha explicado para la ejecución sin ROS en este mismo manual, pero seleccionado el controlador llamado **ROS** que viene por defecto con el programa Webots.

5. **Poner en marcha la simulación.** Una vez asignado el controlador, el usuario debe poner en marcha la simulación mediante el botón correspondiente a la opción ***Run the simulation in real time*** de igual manera que se ha explicado para la ejecución sin ROS en este mismo manual.

```
INFO: ros: Starting: ""/home/gutmor/webots/resources/projects/default/controllers/ros/ros""
[ros] Robot's unique name is carrito_7165_gutmor_VirtualBox
[ros] The controller is now connected to the ros master
```

Figura 82: Mensaje de conexión del controlador de ROS al Nodo Maestro

A.3. Modificar la simulación

Existe la posibilidad de que la intención del usuario no sea únicamente la de ejecutar la simulación ya existente si no también la de modificar elementos dentro de esta o incluso la de crear una nueva simulación. En esta sección del manual se explica de forma sencilla como realizar las principales modificaciones de la simulación y como usar sus elementos en simulaciones nuevas.

A.3.1. Modificar puntos de ruta

Una de las posibles modificaciones que puede querer llevar a cabo el usuario es la de modificar, eliminar o añadir puntos de ruta dentro de la simulación. Para realizar estas operaciones con los puntos de ruta, el usuario debe seguir los siguientes pasos:

1. **Añadir, eliminar o modificar los vectores de coordenadas del punto de ruta en el array de puntos de ruta**

```
// El siguiente array guarda las coodenadas
// en la simulación del campus
static Vector vectores [nDePuntos] = {
{-21.0372, 2.85399, 0}, //P_0
{-24.0101, 10.2883, 1}, //P_1
{-21.0462, 18.4587, 2}, //P_2
{-18.3682, 25.0631, 3}, //P_3
{-10.3588, 21.941, 4}, //P_4
{-3.89893, 19.2597, 5}, //P_5
{-16.4515, 7.02719, 6}, //P_6
{-10.0137, 5.14309, 7}, //P_7
{-0.559824, 1.6976, 8}, //P_8
{-2.34788, -3.63906, 9}, //P_9
{4.02234, 0.0927675, 10}, //P_10
{1.71222, 7.09127, 11}, //P_11
{4.78771, 14.8878, 12}, //P_12
{-15.4077, -9.98163, 13}, //Benet_13
{-4.06377, -5.80206 ,14}, //Betancourt_14
{7.68691, -1.54001, 15}, //Quevedo_Soler_15
{4.35543, 6.26815, 16}, //Biblioteca_16
{-6.52055, 13.1392, 17}, //Sabatini_17
};
```

Figura 83: Array de vectores con las coordenadas de los puntos de ruta

2. **Añadir, eliminar o modificar los puntos de ruta, incluyendo sus adyacencias con otros puntos de ruta.**

```
// A continuación se crean los puntos de ruta con sus coordenadas, puntos adyacentes y pos
puntoDeRuta P_0 = {vectores[0], {vectores[13],vectores[1]},0};
puntoDeRuta P_1 = {vectores[1], {vectores[0],vectores[2],vectores[6]},1};
puntoDeRuta P_2 = {vectores[2], {vectores[1], vectores[3]},2};
puntoDeRuta P_3 = {vectores[3], {vectores[2], vectores[4]},3};
puntoDeRuta P_4 = {vectores[4], {vectores[3], vectores[5]},4};
puntoDeRuta P_5 = {vectores[5], {vectores[4], vectores[17]},5};
puntoDeRuta p_6 = {vectores[6], {vectores[1], vectores[7]},6};
puntoDeRuta p_7 = {vectores[7], {vectores[6], vectores[17], vectores [8]},7};
puntoDeRuta p_8 = {vectores[8], {vectores[7], vectores[11], vectores[9], vectores[10]},8};
puntoDeRuta p_9 = {vectores[9], {vectores[8], vectores[14]},9};
puntoDeRuta p_10 = {vectores[10], {vectores[8], vectores[15]},10};
puntoDeRuta p_11 = {vectores[11], {vectores[8], vectores[16]},11};
puntoDeRuta P_12 = {vectores[12], {vectores [11]},12};
puntoDeRuta Benet_13 = {vectores[13], {vectores[0]},13};
puntoDeRuta Betancourt_14 = {vectores[14], {vectores[9]},14};
puntoDeRuta Quevedo_Soler_15 = {vectores[15],{vectores[9], vectores[10]},15};
puntoDeRuta Biblioteca_16 = {vectores[16], {vectores[11]},16};
puntoDeRuta Sabatini_17 = {vectores[17], {vectores[5], vectores[7]},17};
```

Figura 84: Declaración de los puntos de ruta

3. **Añadir, eliminar o modificar los puntos de ruta referenciados dentro del array de puntos de ruta.**

```
// Este array de puntos de ruta sirve para hacer mas sencilla la busqueda y uso de estos
static puntoDeRuta puntos [nDePuntos] = {
P_0,
P_1,
P_2,
P_3,
P_4,
P_5,
p_6,
p_7,
p_8,
p_9,
p_10,
p_11,
P_12,
Benet_13,
Betancourt_14,
Quevedo_Soler_15,
Biblioteca_16,
Sabatini_17,
};
```

Figura 85: Array de puntos de ruta

A.3.2. Modificar posición del vehículo autónomo

Para modificar la posición de partida del vehículo autónomo dentro de la simulación pueden usarse dos métodos diferentes:

1. **Manipular directamente sus propiedades.** Las coordenadas y orientación del vehículo autónomo pueden modificarse a voluntad en el árbol de escena mediante las propiedades "***translation***" y "***rotation***".

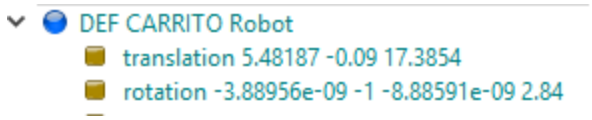


Figura 86: Cambio de posición del vehículo autónomo mediante sus coordenadas

2. **Arrastrar el vehículo en la representación 3D.** Todos los elementos de la simulación pueden ser trasladados si se seleccionan con el ratón y se arrastran dentro de esta en la ventana en la que se muestra la representación en tres dimensiones de la simulación.

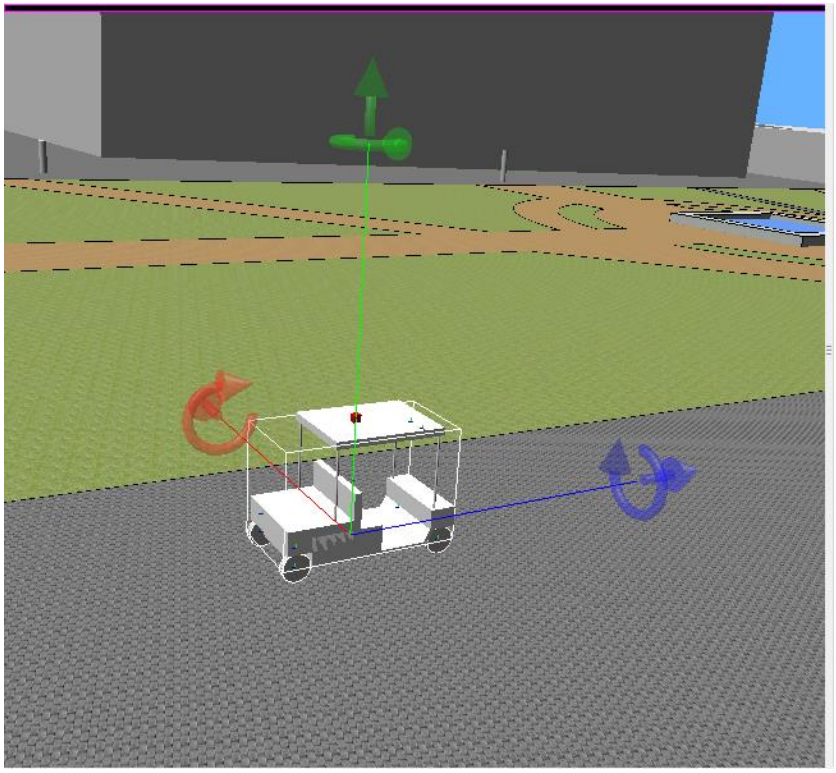


Figura 87: Cambio de posición del vehículo autónomo mediante su representación gráfica

A.3.3. Usar el vehículo autónomo en otro entorno simulado

Es posible usar el vehículo autónomo definido en esta simulación en otra simulación creada por el usuario. Para ello, el usuario debe seguir los pasos siguientes:

1. **Exportar el modelo del vehículo autónomo.** Para poder usar el vehículo en otra simulación, el usuario debe primero exportarlo desde la simulación del proyecto. Para ello, debe seleccionar el nodo del vehículo autónomo dentro del árbol de la simulación y elegir la opción "***Export…"***. Una vez elegida la ruta donde lo va a exportar, se generará un archivo con extensión .wbo (*Webots Object)* que contiene el modelo del vehículo autónomo.

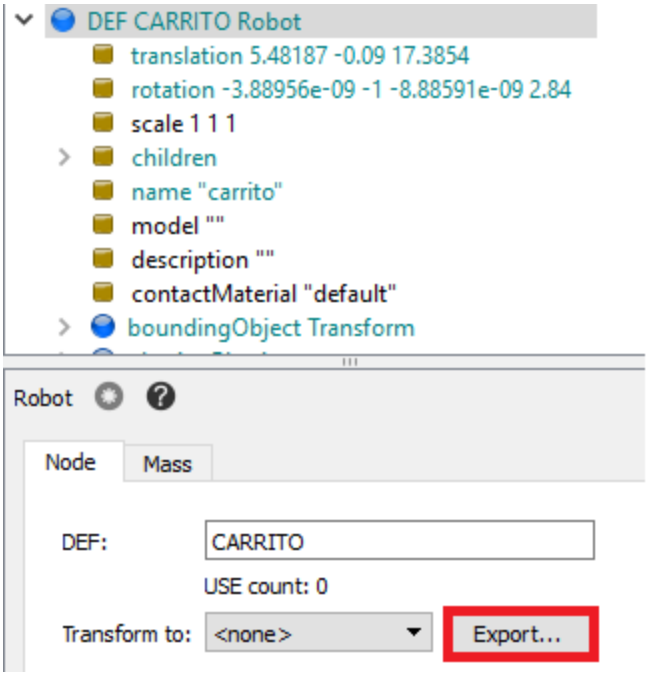


Figura 88: Exportar vehículo autónomo de la simulación

2. **Importar el vehículo en el nuevo entorno simulado.** Para poder usar el vehículo en el nuevo entorno, el usuario tendrá que seleccionar la opción de añadir nuevo nodo (un símbolo + en la ventana principal de Webots) y usar la opción ***"Import..."*** en el menú que se desplegará. Tras seleccionar la opción de importar, únicamente debe seleccionar el archivo con el modelo del carrito creado en el paso anterior.

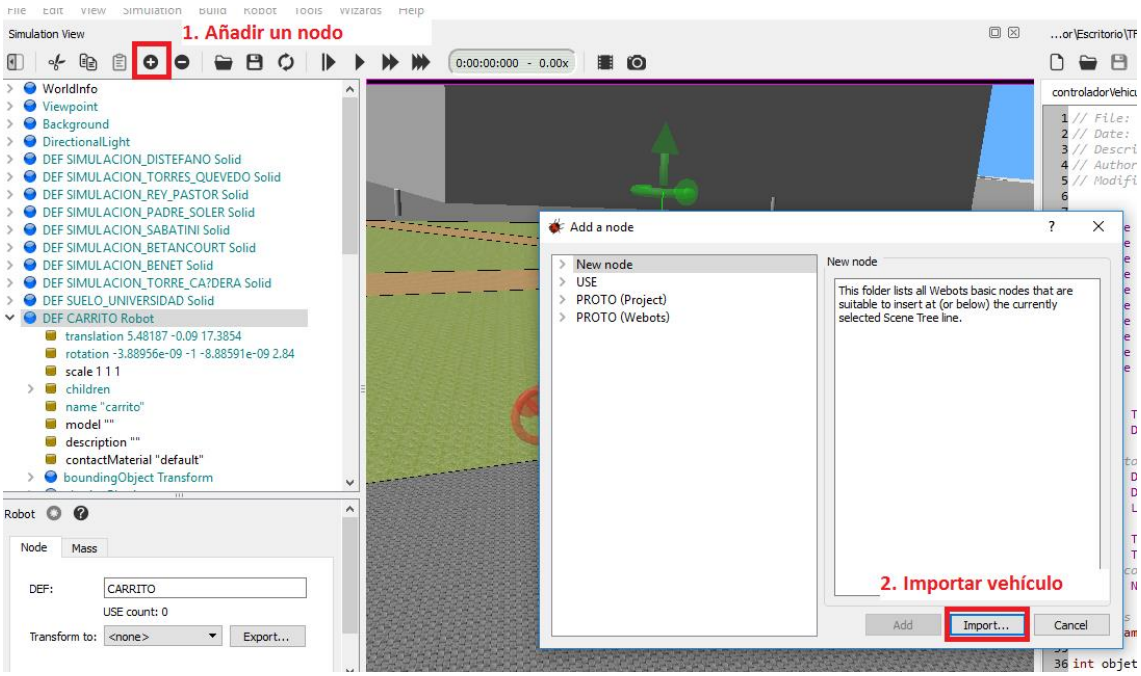


Figura 89: Importar el vehículo autónomo dentro de una nueva simulación

A.3.4. Usar el controlador en otro vehículo

Es posible utilizar el controlador del vehículo autónomo implementado en este proyecto con otro vehículo autónomo creado por el usuario. Para ello, se tienen que cumplir las siguientes condiciones:

- **El nuevo vehículo debe tener los mismos dispositivos básicos.** El vehículo creado por el usuario debe tener al menos un láser SICK, un dispositivo GPS, un dispositivo brújula, dos dispositivos de tipo motor correspondientes a las ruedas motrices y dos dispositivos motor correspondientes a las ruedas motrices.
- **Deben modificarse las medidas del vehículo dentro del controlador.** Al comienzo de la implementación del controlador, vienen definidas tres variables que se corresponden con medidas del vehículo autónomo simulado. Para trabajar con otro vehículo, deben introducirse sus medidas, cambiando los valores de estas variables. Las variables corresponden al diámetro de las ruedas, la longitud del eje (distancia entre las ruedas izquierdas y derechas) y la distancia de los ejes (distancia entre las ruedas delanteras y traseras).

```
#define DIAMETRO_RUEDAS 0.072
#define DISTANCIA_EJES 0.4
#define LONGITUD_EJES 0.212
```

Figura 90: Variables que identifican las medidas del vehículo autónomo dentro del controlador

- **Los nombres de los dispositivos deben coincidir.** Los nombres de los dispositivos del vehículo autónomo deben coincidir con los que aparecen en el código del controlador. El usuario puede, o bien modificar el nombre de los dispositivos de su vehículo, o bien incluir el nombre de sus dispositivos en el código del controlador.

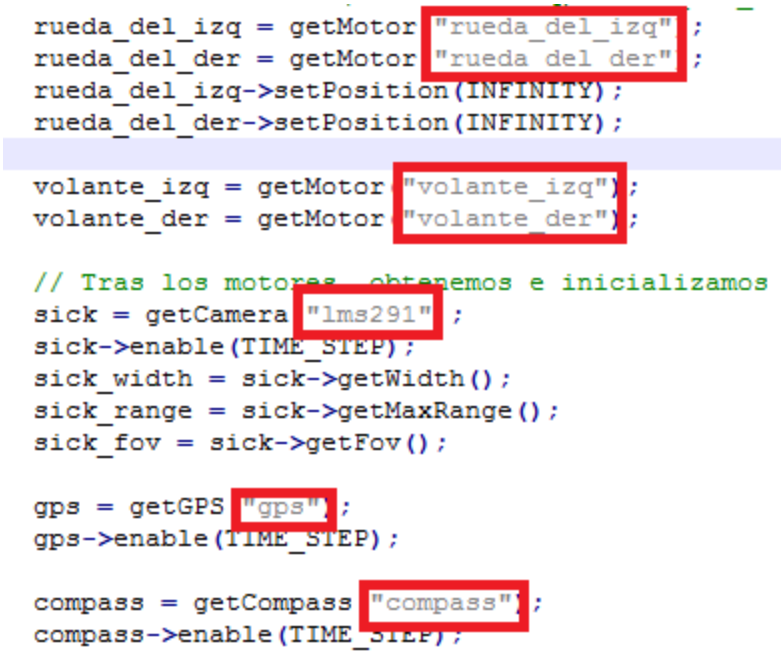


Figura 91: Nombres que identifican los dispositivos del vehículo en el controlador

Anexo B. Modelos del campus

A continuación, se muestran las simulaciones realizadas de los edificios el Campus de la Escuela Politécnica Superior de la Universidad Carlos III de Madrid.

B.1. San Agustín de Betancourt

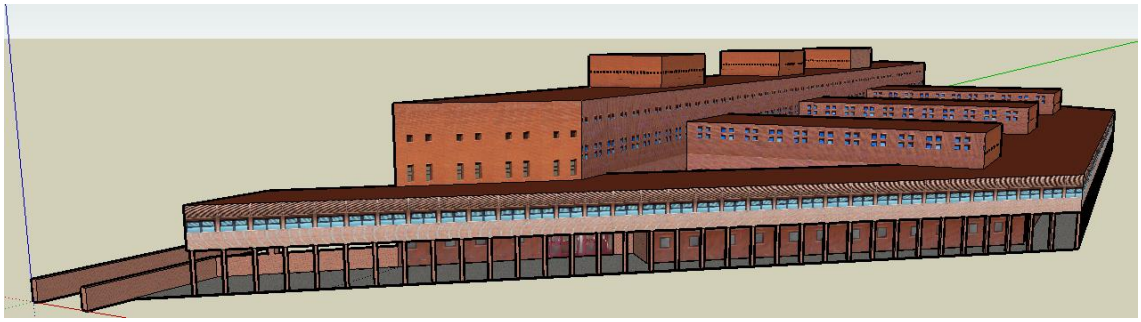


Figura 92: Fachada (1) Betancourt

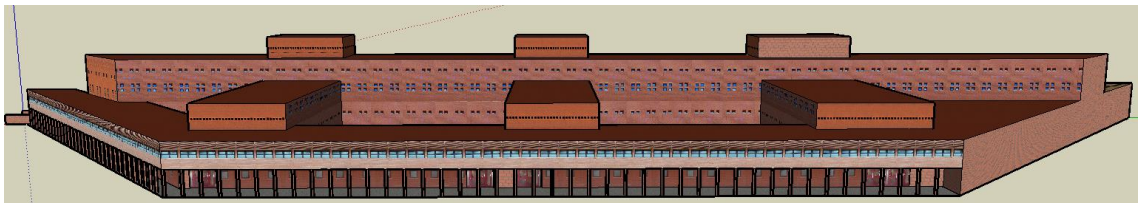


Figura 93: Fachada (2) Betancourt

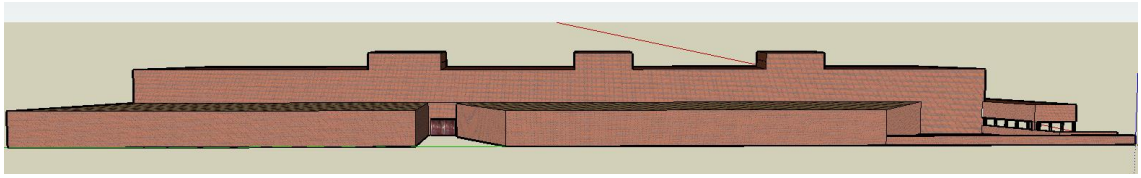


Figura 94: Fachada (3) Betancourt

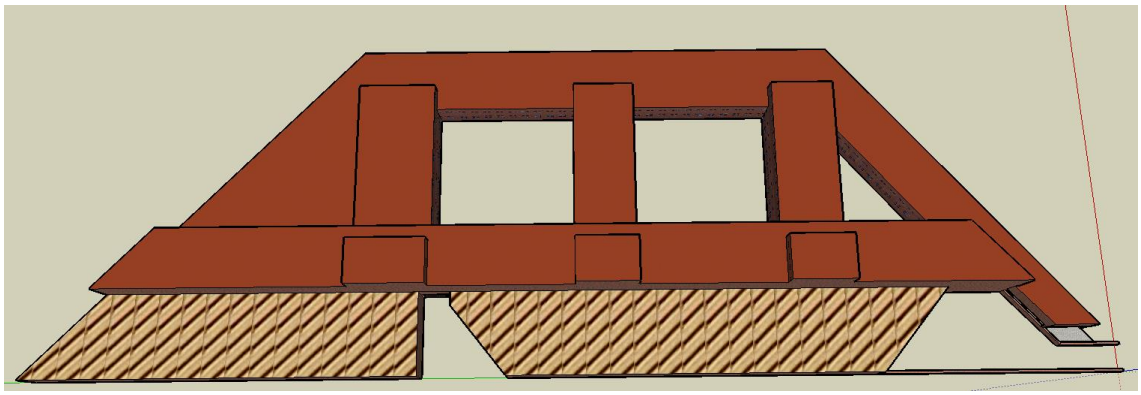


Figura 95: Plano cenital Betancourt

B.2. Sabatini

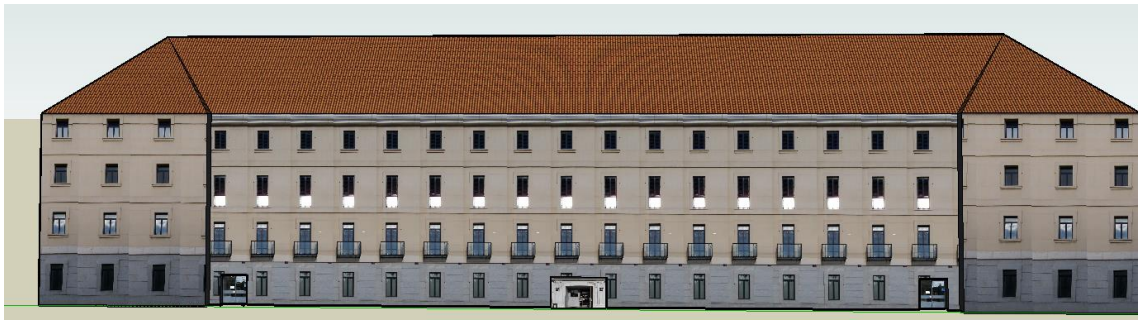


Figura 96: Fachada (1) Sabatini

Figura 97: Fachada (2) Sabatini

El modelado de las restantes fachadas del edificio Sabatini son iguales que la fachada 2 (ver Figura 97).

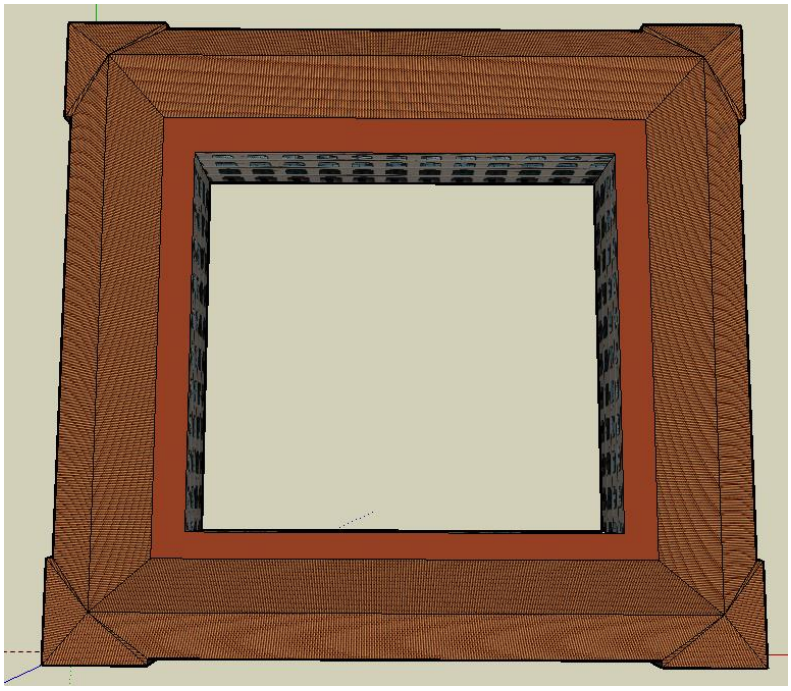


Figura 98: Plano cenital Sabatini

B.3. Rey Pastor

Figura 99: Fachada (1) Rey Pastor

Figura 100: Fachada (2) Rey Pastor

El modelado de las restantes fachadas del edificio Rey Pastor son iguales que la fachada 2 (ver Figura 100).

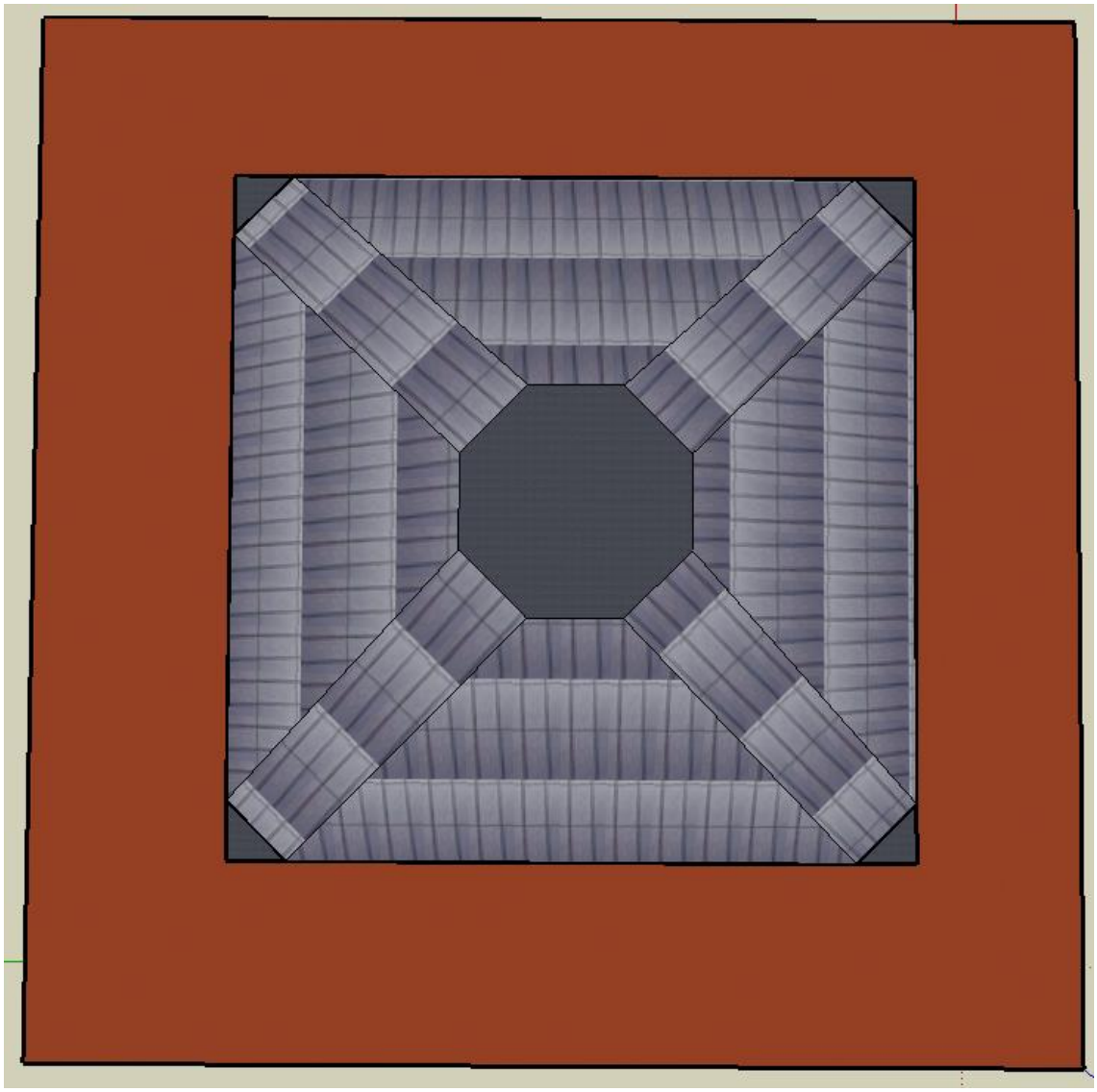


Figura 101: Plano cenital Rey Pastor

B.4. Torres Quevedo

Figura 102: Fachada (1) Torres Quevedo

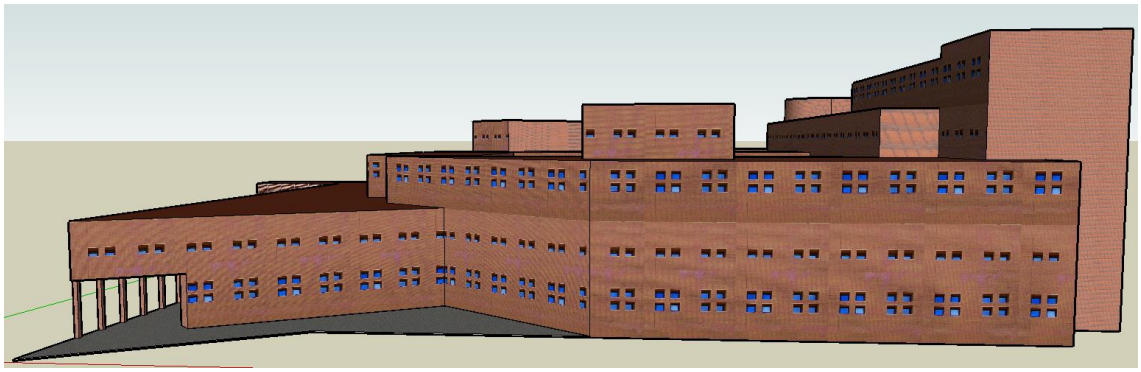


Figura 103: Fachada (2) Torres Quevedo

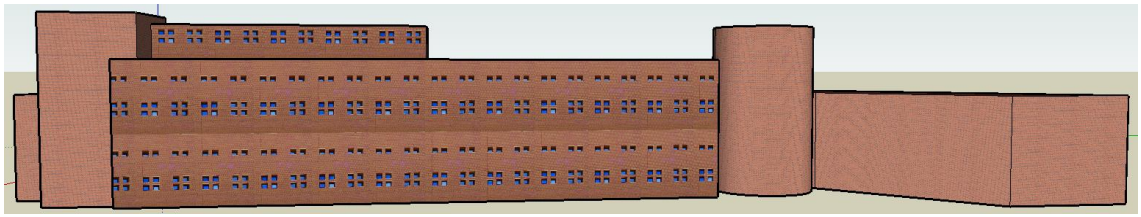


Figura 104: Fachada (3) Torres Quevedo

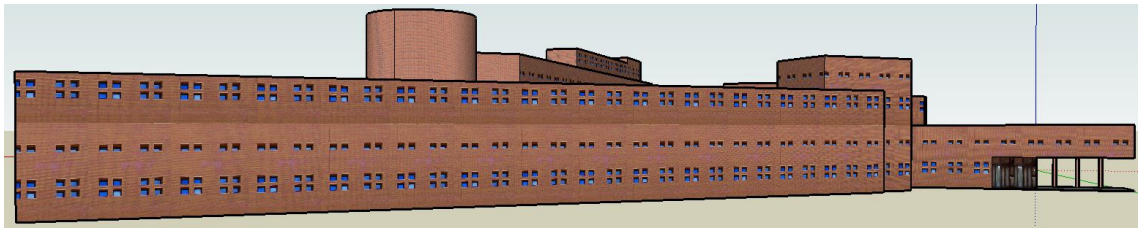


Figura 105: Fachada (4) Torres Quevedo

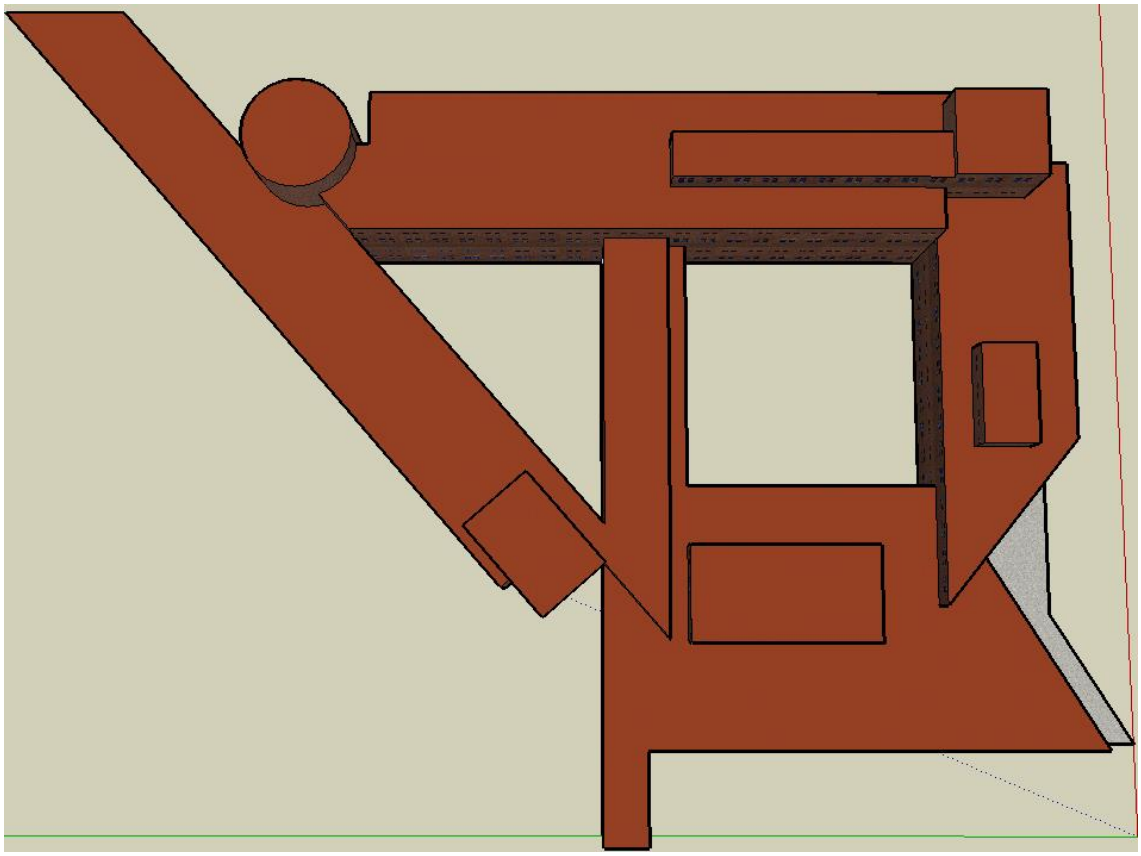


Figura 106: Plano cenital Torres Quevedo

B.5. Padre Soler

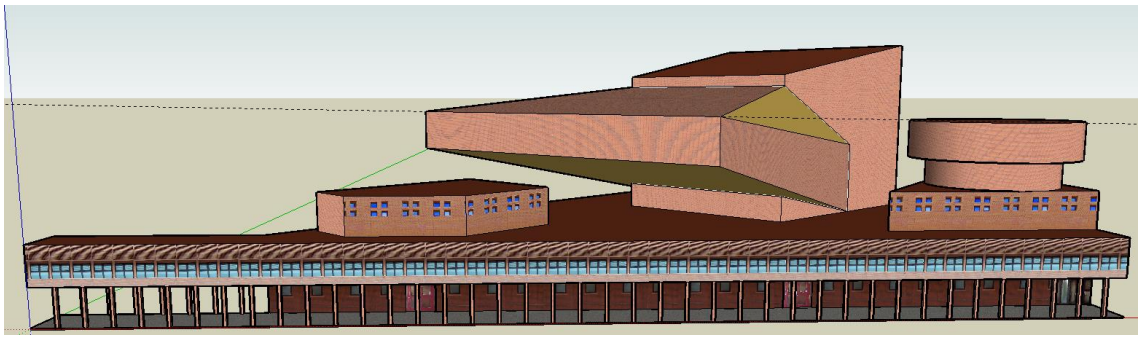


Figura 107 : Fachada (1) Padre Soler

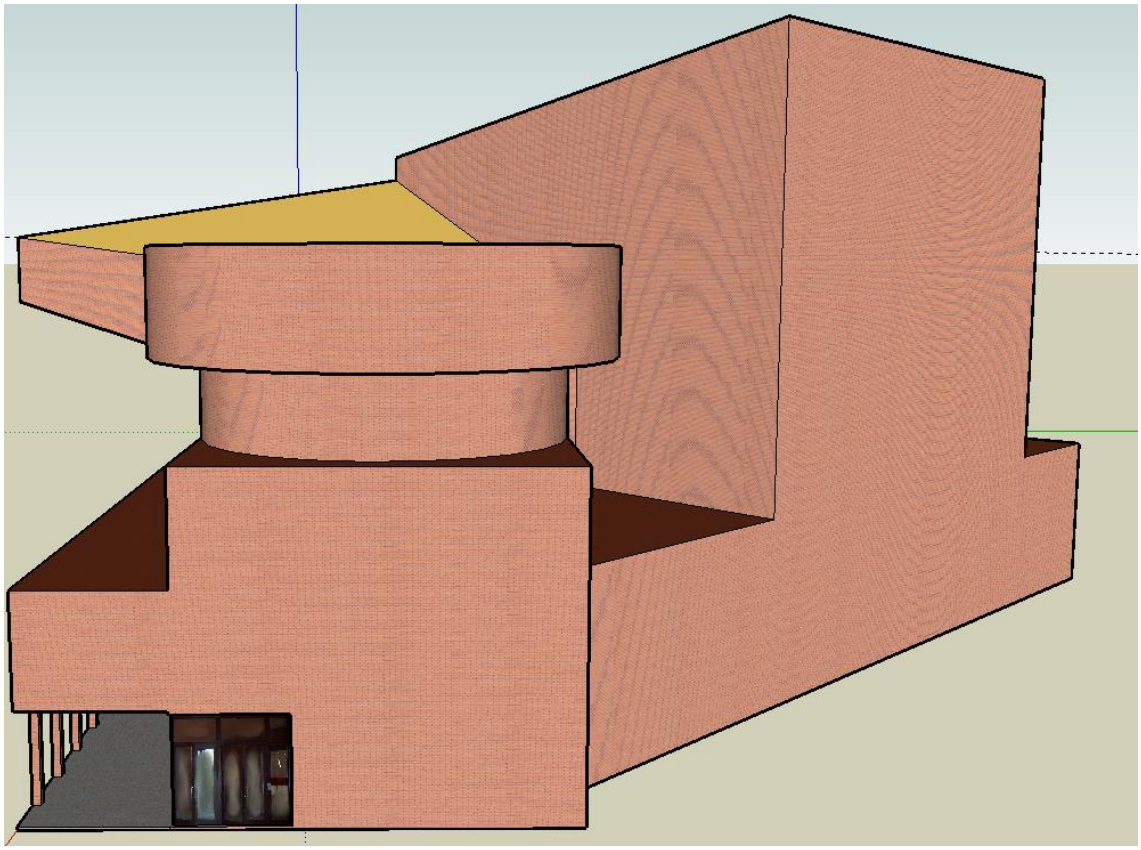


Figura 108 : Fachada (2) Padre Soler

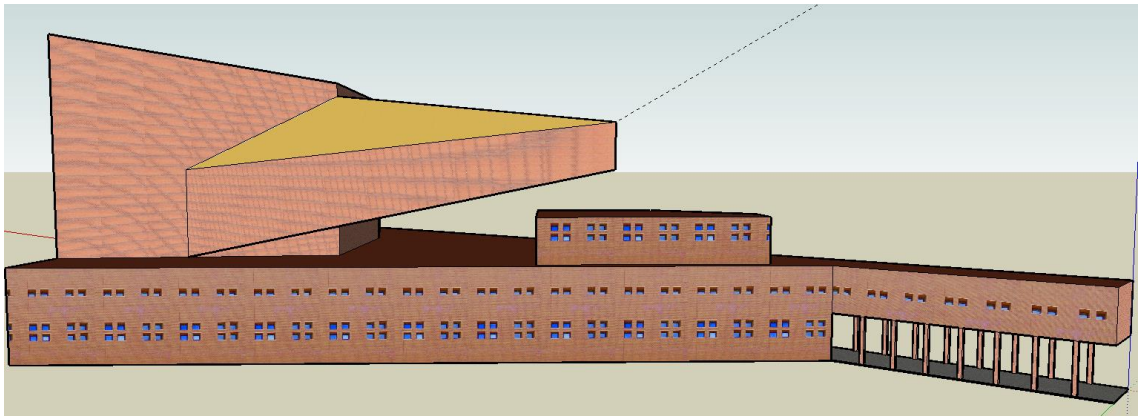


Figura 109 : Fachada (3) Padre Soler

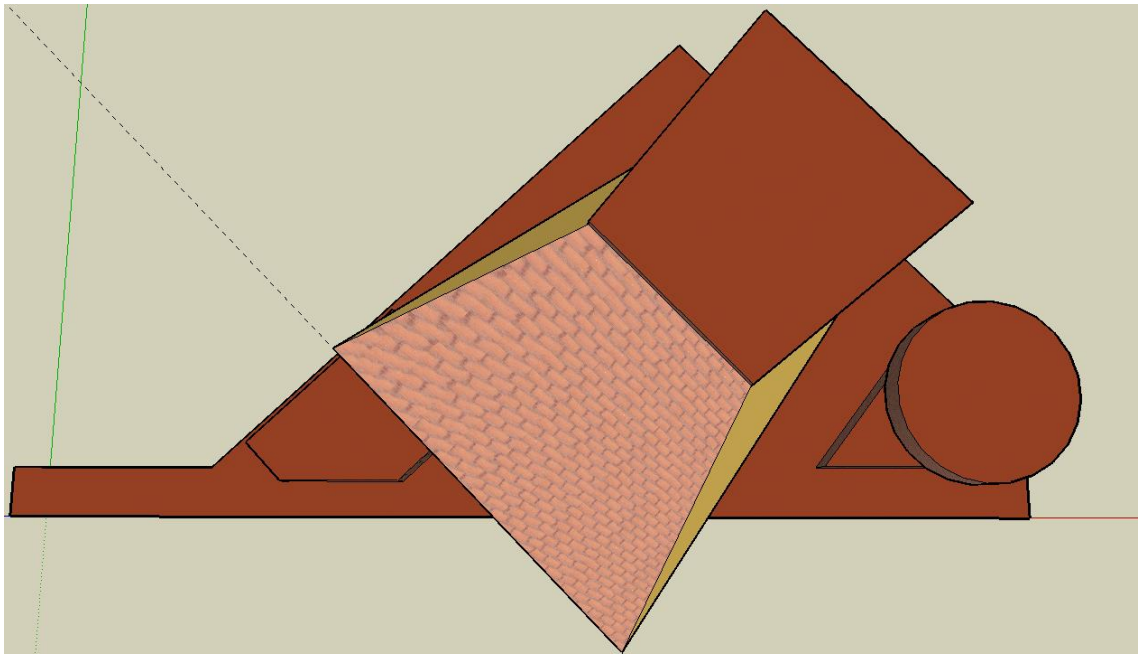


Figura 110: Plano cenital Padre Soler

B.6. Alfredo Di Stefano

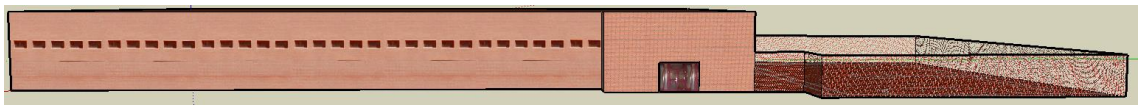


Figura 111 : Fachada (1) Alfredo Di Stefano

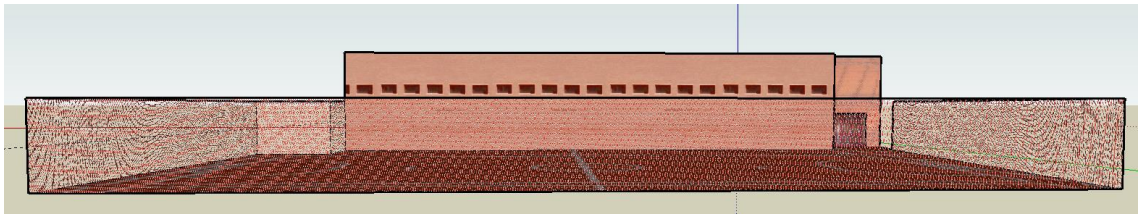


Figura 112 : Fachada (2) Alfredo Di Stefano

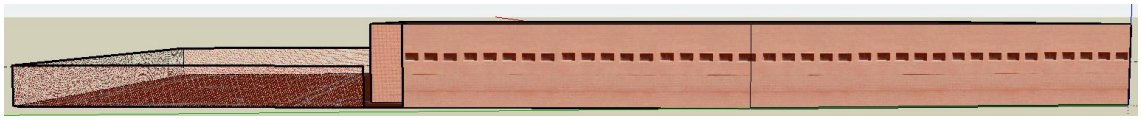


Figura 113 : Fachada (3) Alfredo Di Stefano

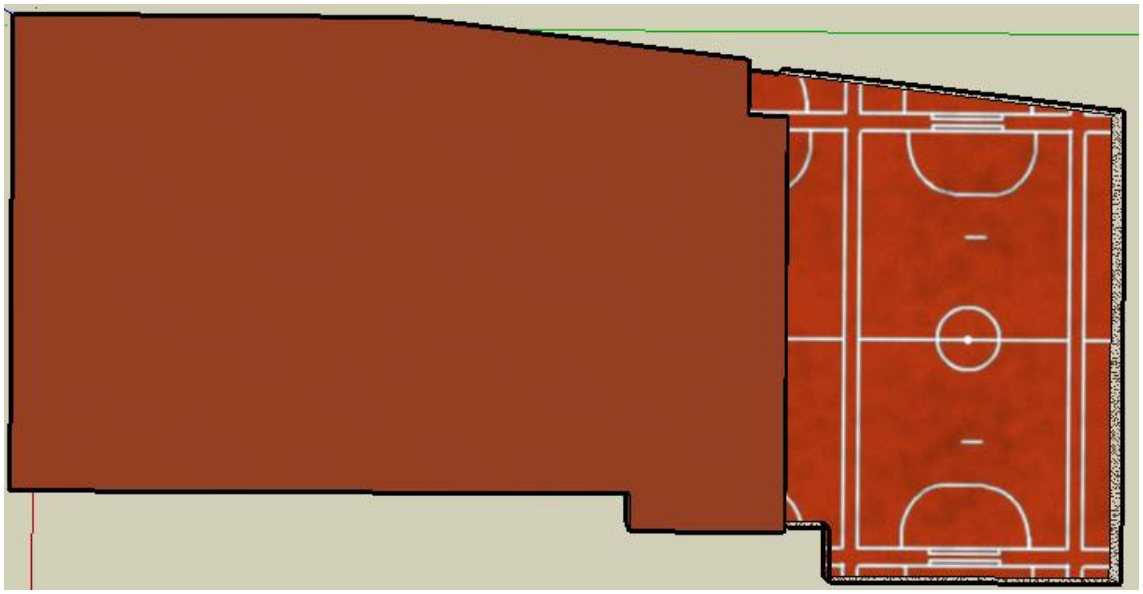


Figura 114: Plano cenital Alfredo Di Stefano

B.7. Juan Benet

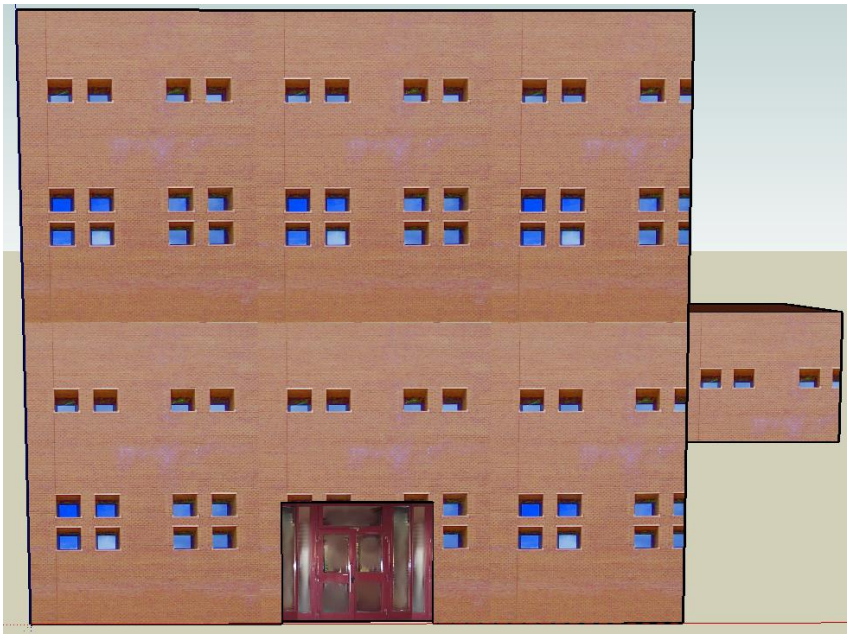


Figura 115 : Fachada (1) Juan Benet

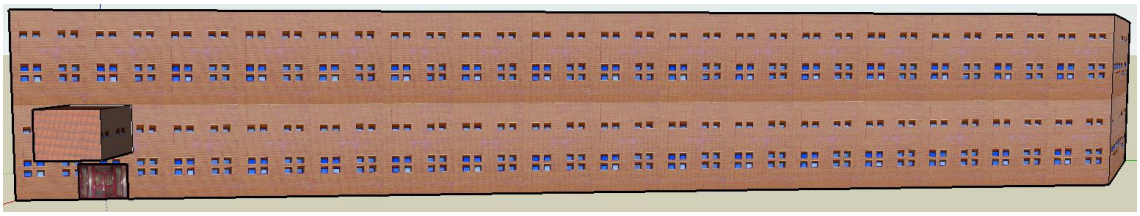


Figura 116 : Fachada (2) Juan Benet

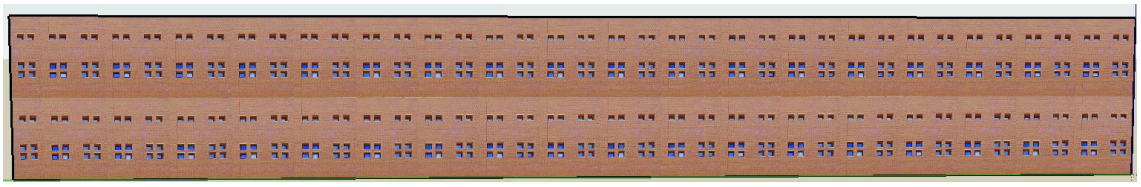


Figura 117 : Fachada (3) Juan Benet

Figura 118: Plano cenital Juan Benet

B.8. Torre de la Caldera

Figura 119: Fachada Torre de la Caldera

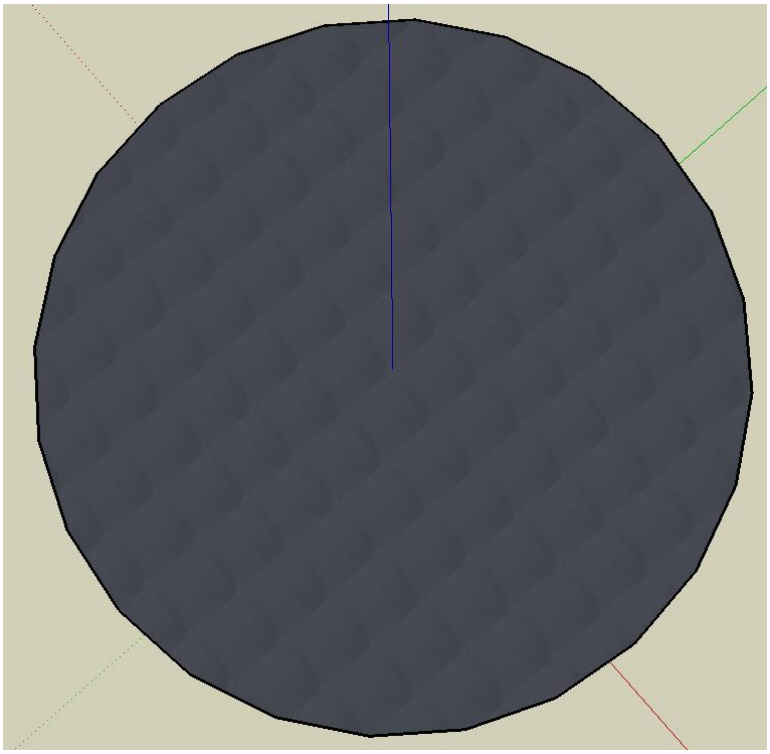


Figura 120: Plano cenital Torre de la Caldera

Anexo C. Resumen en Inglés.

En este anexo se realiza una exposición en inglés de los apartados de "Introducción", "Conclusiones y trabajos futuros" y "Resultados y experimentación".

C.1. Introduction

This year 2016 marks the 130th anniversary of the beginning of the automobile age. In 1886 the engineer Karl Friedrich Benz patented the first internal combustion motor vehicle in history, which he named Benz Patent-Motorwagen.

A few years later, in 1908, the businessman Henry Ford, father of the modern assembly lines, began the series production of automobile in an assembly line, another milestone in automotive history that allowed internal combustion vehicles gradually becoming into the popular means of transport they are today.

C.1.1. Motivation

In the last 130 years, the automobile has evolved by leaps and bounds. This evolution has not focused only on getting faster and more powerful vehicles, but also safer.

With the widespread use of the automobile, also appeared one of the greatest tragedies of the modern era, deaths by traffic accident. According to a report by the WHO (World Health Organization) in 2013 on road safety [1], about 1.25 million people die each year on traffic accidents and between 20 and 50 million are injured. These data are alarming and explain what is the current trend in automotive design; the development of security systems and driver assistance.

These systems of driver assistance, also known as ADAS (acronym of Advanced Driving Assistance Systems), aim to eliminate or, at least, reduce the influence of human errors during driving, which are the cause of most traffic accidents. But he current trend is not only better assistance systems, also completely autonomous vehicles are being developed, capable of circulate without any driver intervention, eliminating completely the human factor in driving, with the advantages and disadvantages it entail.

The objective of this work is to contribute to the development of autonomous vehicles, in order to make driving a safer activity.

C.1.2. Description of the problem

Implement a controller, ie a computer program to interact with a vehicle to run its operation, has a major problem, it requires a high expenditure of resources, both temporal and economic when testing for check the correct operation of the said controller. In addition to these resource problems, there are security problems, since due to the nature of the test to be carried out with the vehicle these can't be done directly on public roads because of the risk they pose to both the vehicle itself as for the personnel performing the test, in addition to other users of the road where they are made.

Therefore, to perform these test is required digitally simulate both the vehicle and the environment in which it will operate to perform they in a safely, controlled and repeatable way.

C.1.3. Scope of the project

The main objective of this work is to develop a controller under the ROS (Robotic Operating System) framework [2] for an autonomous vehicle in a simulated environment that will recreate the campus of the campus of the Escuela Politécnica of the Universidad Carlos III de Madrid.

The specific objectives of the project are:

- Develop the simulated environment of the campus of the Escuela Politécnica of the Universidad Carlos III de Madrid.
- Develop a simulated model of the autonomous vehicle to be used.
- Implement a controller that operate the autonomous vehicle in the simulated campus environment in the C++ programming lenguage.
- Implement and validate the previous controller under the ROS infrastructure.

C.1.4. Memory organization

This section describes the organization of this document, briefly exposing the content of each chapter.

In Chapter 1 an introduction to the project is performed. This introduction includes the motivation for the project, the description of the problem, the scope of the proyect and, finally, the organization corresponding to the project report document itself.

In Chapter 2 an analysis of the state of the art, both as regards autonomous vehicles, such as robotics and driving simulators is exposed.

In Chapter 3 the analysis and design of the system that has been implemented is detailed. This analysis includes a description of the rules and restrictions under which the project is realized, its operational environment, and the specification of its requirements, both functional and non-functional, and use cases.

In Chapter 4 the process and results of the elements that form the simulation is shown. The simulated elements shown are the campus of the Escuela Politécnica of the Universidad Carlos III de Madrid and the autonomous vehicle.

In Chapter 5 a description of the implementation of the controller developed in this project is done. This description includes the controller implementation in C ++ and its version in ROS. Finally the results of the experimentation carried out with the controller are shown.

In Chapter 6, work planning and project budgets are defined. The methodology used for project development, the work plan elaborated and the budgets broken down into personnel costs, material costs and total project cost project are explained.

In Chapter 7 the conclusions, both technical and personal, the work done and future lines of work that can be followed from this project and possible improvements that can be made on it are exposed.

As a final part of the document, bibliography and references as well as the annexes are included.

C.2. Conclusions and future Works

In this section it are exposed the conclusions drawn from the completion of this project, both technical conclusions as personal conclusions and future lines of work that can be taken from the project made.

C.2.1. Technical conclusions

Once the project is completed, it can be concluded that all the objectives set at the beginning of this have been completed in a satisfactory way.

The first objective which has been completed has been the modeling of a simulated environment of the campus of the Escuela Politécnica of the Universidad Carlos III de Madrid that largely resembles the actual campus. Similarly it has been satisfactorily performed the modeling of a fully functional robot in the form of an autonomous vehicle, which resembles the real autonomous vehicles owned by the Universidad Carlos III de Madrid.

Another of the objectives that have been carried out satisfactorily is the implementation of a controller for an autonomous vehicle capable of driving on the simulated environment of the campus of the Escuela Politécnica of the Universidad Carlos III de Madrid dodging the obstacles that finds on its way as it heads to a specific point within the campus.

The last goal proposed and which has also been achieved successfully is to adapt the controller of the vehicle to the framework of ROS, thus allowing future work on the controller and its application to a real vehicle will be much faster and easier, and that the simulation carried out can be transferred to the real world in a direct way.

C.2.2. Personal conclusions and knowledge adquire

From a personal point of view, the conclusions drawn at the end of this project are very positive. Having finished this project so successfully is a great end to the six years I spent at this university and the achievement, at last, of the degree in Computer Engineering.

During these six years I had never had to deal with a project of this magnitude without more help than the guidance of my tutor and my own research and learning capabilities. This experience has made me be part of first hand which is to carry out the realization of a research project within a working group as is the CAOS group of the Universidad Carlos III, and take experience for future projects, both in college and future jobs.

But I have not only gained work experience, I have also learned to use tools that, before this project had not heard of such as SketchUp, Webots or ROS. The experience gained with these tools, as well as the C ++ programming language, are an extra value that led me of the realization of this project.

C.2.3. Future lines of work

The work done in this project lays the groundwork for different future lines of work. The main line of work that can be follow from the work developed in this project is to implement the controller of the autonomous vehicle in a physical autonomous vehicle since, as has been commented above, having implemented the controller in the framework ROS makes the process of translate the controller from the simulated vehicle to the actual vehicle a direct process.

Other future research that can be developed from work done is work to improve simulation and refinement work on the autonomous vehicle controller. Examples of these lines of work are:

- **Reshape campus buildings**. The level of detail with which campus buildings are modeled is so high that causes the execution of the simulation be too slow due to memory overload. A possible future work is remake the models with a lower level of detail to allow a smoother execution of the controller.
- **Add new features**. It is possible that, depending on the needs of future work and the new required functionalities, be necessary to extend the controller for the vehicle to carry out new actions. The controller code is widely commented to be easy to add or remove functionality to the controller.
- **Refine the controller**. It is also posible to refine or improve the operation of the controller by modifying its functions and parameters. The code structure and comments allow its modification to be simple and friendly.

Note also that the controller can be used in any simulated environment, not limited only to the environment of the campus of the Escuela Politécnica of the Universidad Carlos de Madrid, and that by the way in which the controller is implemented would only be necessary enter the waypoints for the new scenario that will be used. Because of this, another possible future work would be to model new environments in which use the controller for the autonomous vehicle. In addition to this, the controller can also be adapted to other vehicles, not just for the one modeling in this project by simply changing the parameters in the code of the controller relating to the vehicle to which it will apply the controller (the space between their axes, the name of its elements in the simulation, etc).

Given the above, it can be concluded that the work done in this project can be used in the future for the simulation of any vehicle (with a structure similar to that used in the project) in any simulated environment, making minor modifications to the implementation carried out, in addition to adapt easily to a real vehicle thanks to its integration with ROS.

C.3. Experimentation and results

In this section it is shown the results of different executions of the implemented controller for the autonomous vehicle. The controller settings and the process followed by the controller during its execution would be detailed.

C.3.1. Scenario 1

In this execution of the controller the waypoint corresponding to the Rey Pastor building has been established as destination point for the autonomous vehicle. The initial position of the vehicle is established across the campus, next to the Sabatini building. The initial setup of the experiment is shown in Figura 121.

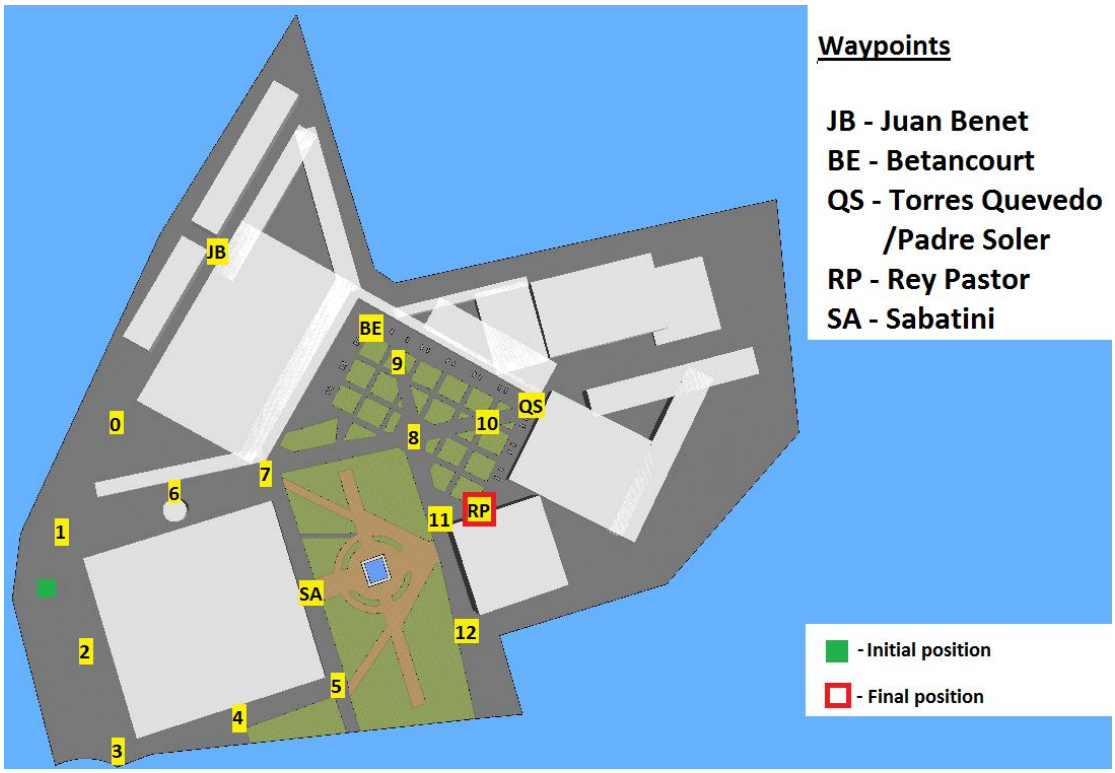


Figura 121: Initial configuration of Scenario 1

When the controller is executed with this configuration of the scenario, the first step is to find the initial waypoint by **`puntoInicial`** function. This function searches which is the nearest waypoint to the point which currently holds the vehicle and sets the waypoint as the first point of the route. In this case, the chosen point is the waypoint number 1.

Once established the initial waypoint of the route, the **`busquedaDeCamino`** method is executed until it reaches the target waypoint. This method works with a heuristic function as follows:

$$f(n) = g(n) + h(n)$$

Where:

- **g(n):** is the distance between the waypoint where the vehicle is now and the waypoint that the algorithm is evaluating as a possible point to go.
- **h(n):** is the distance between the waypoint that the algorithm is evaluating as a possible point to go and the destination.

The pathfinding algorithm to choose the next waypoint to go is as follows:

1. The heuristic value of all adjacent points to the current one is checked and the one that has a lower heuristic value is selected as destination.
2. At this point there are two possibilities:
    a. If there is a valid adjacent point, the waypoint is saved in the list of waypoints to follow.
    b. If the point where the algorithm is now has not any valid adjacent waypoints (because their adjacent points have been visited or discarded), it means that this path is incorrect, since it is not possible to reach the destination. If this is the case, this waypoint is removed from the route, adds to a list of discarded and set as current point the previous point from the route.

This algorithm is repeated until the current waypoint coincides with the target waypoint. When this occurs, the vehicle travels across each of the waypoints that have been defined as route.

In this scenario, the chosen route to be followed by the autonomous vehicle is: **1-6-7-8-11-RP.** Figura 122 shows the process followed by the algorithm to calculate the route.

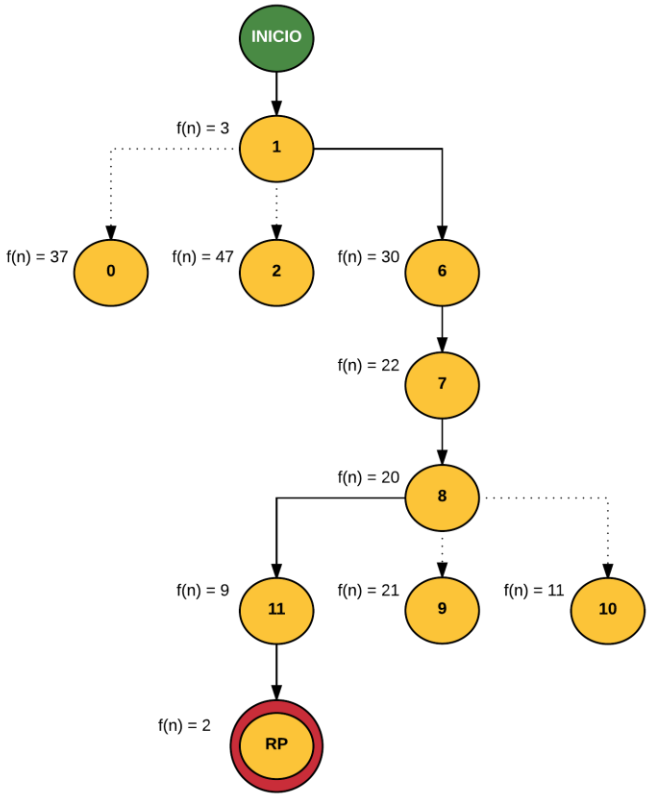


Figura 122: Process of the pathfinding algorithm in Scenario 1

The pathfinding algorithm has chosen a correct path from the starting point to the destination. Following this pathfinding, vehicle movement begins across the campus following the marked path to reach the goal. The route traveled within the simulation is shown in Figura 123.

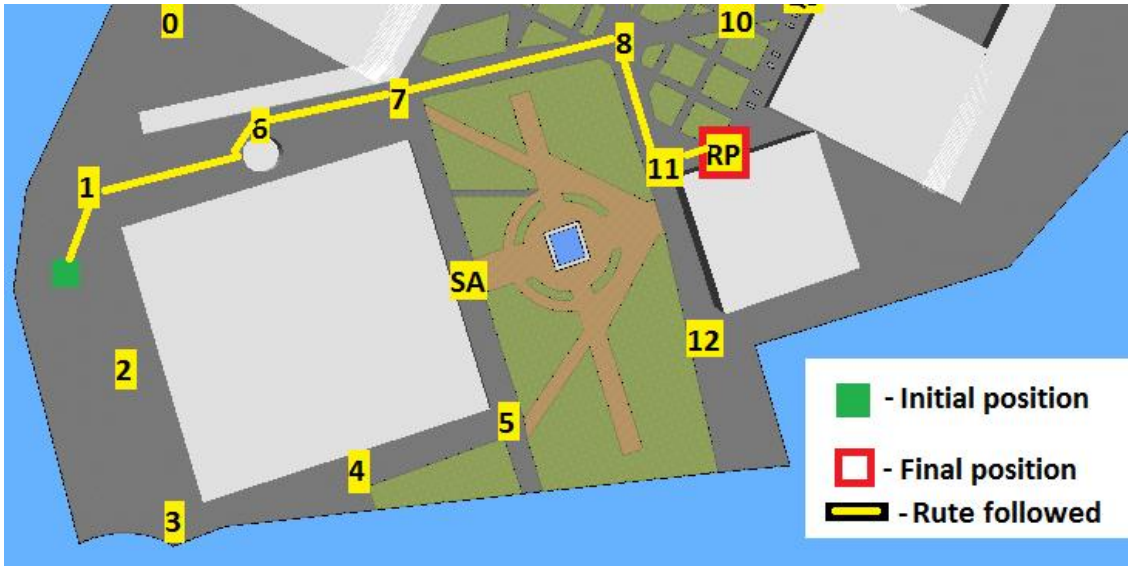


Figura 123: Path followed by the autonomous vehicle in Scenario 1

The autonomous vehicle travels the route established by the pathfinding algorithm correctly. In the course from waypoint 1 to waypoint 6 the autonomous vehicle prevents the collision with the Torre de la Caldera building and goes to its target. The result of the experiments in this scenario it is satisfactory.

C.3.2. Scenario 2

In this execution of the controller the waypoint corresponding to the Rey Pastor building has been established as destination point for the autonomous vehicle. The initial position of the vehicle is established across the campus, next to the Rey Pastor building. Also an obstacle that the autonomous vehicle will have to avoid has been added between waypoint 1 and waypoint 0. The initial setup of the experiment is shown in Figura 124.

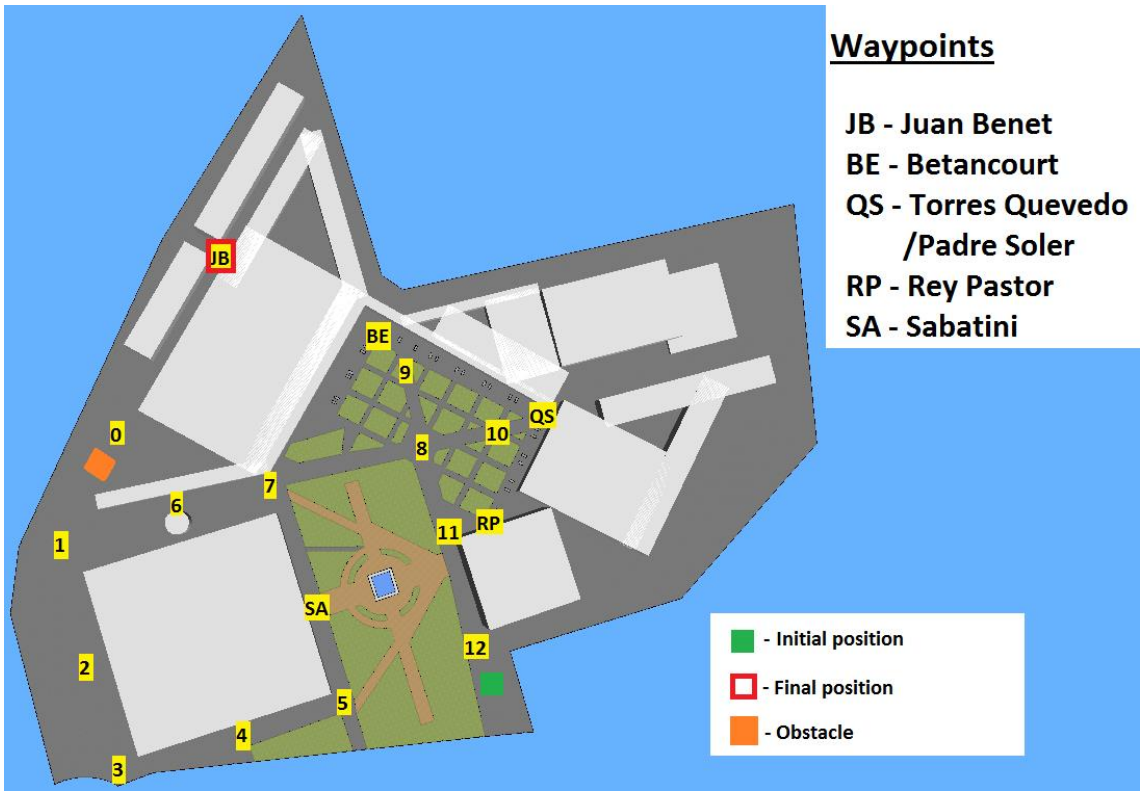


Figura 124: Initial configuration of Scenario 2

In this scenario, the chosen route to be followed by the autonomous vehicle is: **12-11-8-7-6-1-0-JB**. Figura 125 shows the process followed by the algorithm to calculate the route.

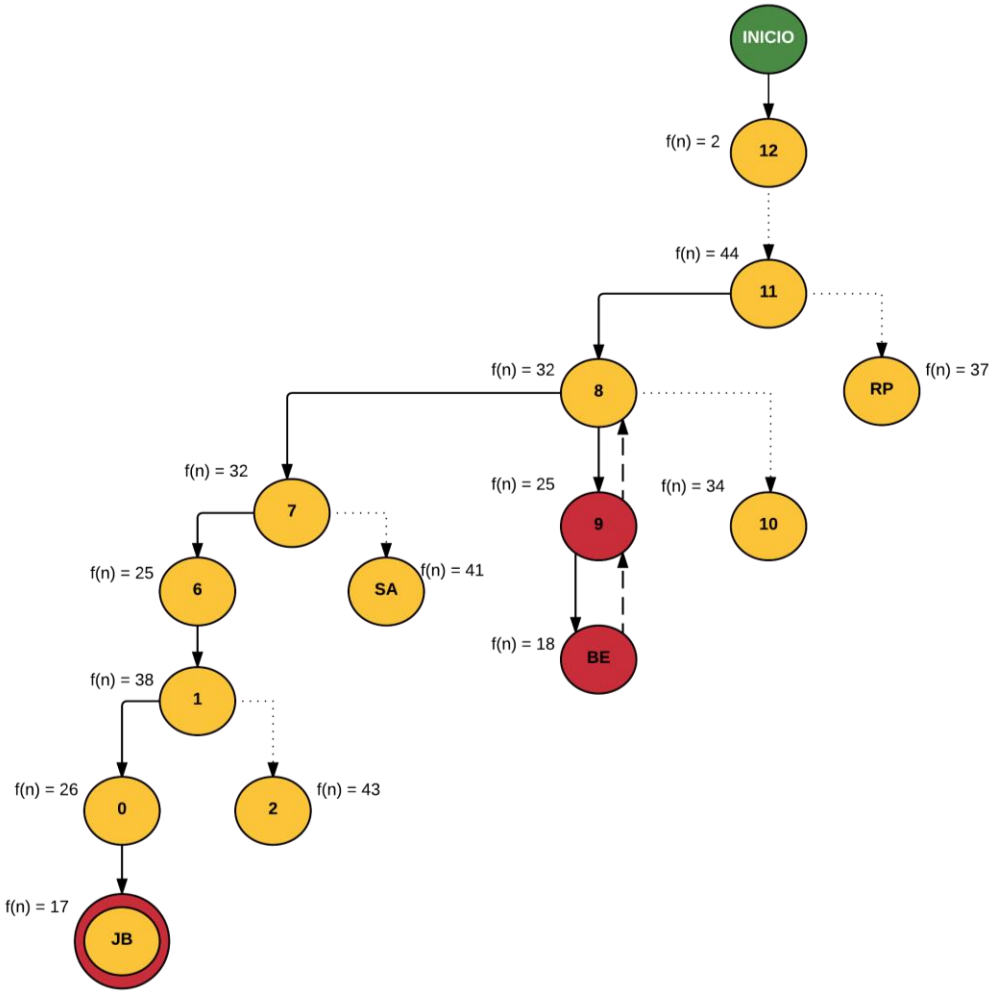


Figura 125: Process of the pathfinding algorithm in Scenario 2

As can be seen in Figura 125, to reach the waypoint 8, in the first instance the algorithm chooses as target the waypoint 9 because the value of their heuristic function is the smallest among the available options. The waypoint 9 has only one adjacent point, corresponding to Betancourt building, so in the next iteration that waypoint is the selected waypoint. At this moment, the waypoint where the algorithm is emplaced has no other adjacent waypoint and is not the destination, therefore the algorithm eliminates this waypoint as possible waypoint to develop a route and returns to the previous one. When it returns to the waypoint 9, this point has no longer any adjacent waypoint due to having eliminated it in the previous step, so this point is removed and the algorithm returns to the waypoint 8, where it continues to develop the path to the end.

The pathfinding algorithm has chosen a correct path from the starting point to the destination. Following this pathfinding, vehicle movement begins across the campus following the marked path to reach the goal. The route traveled within the simulation is shown in Figura 126.

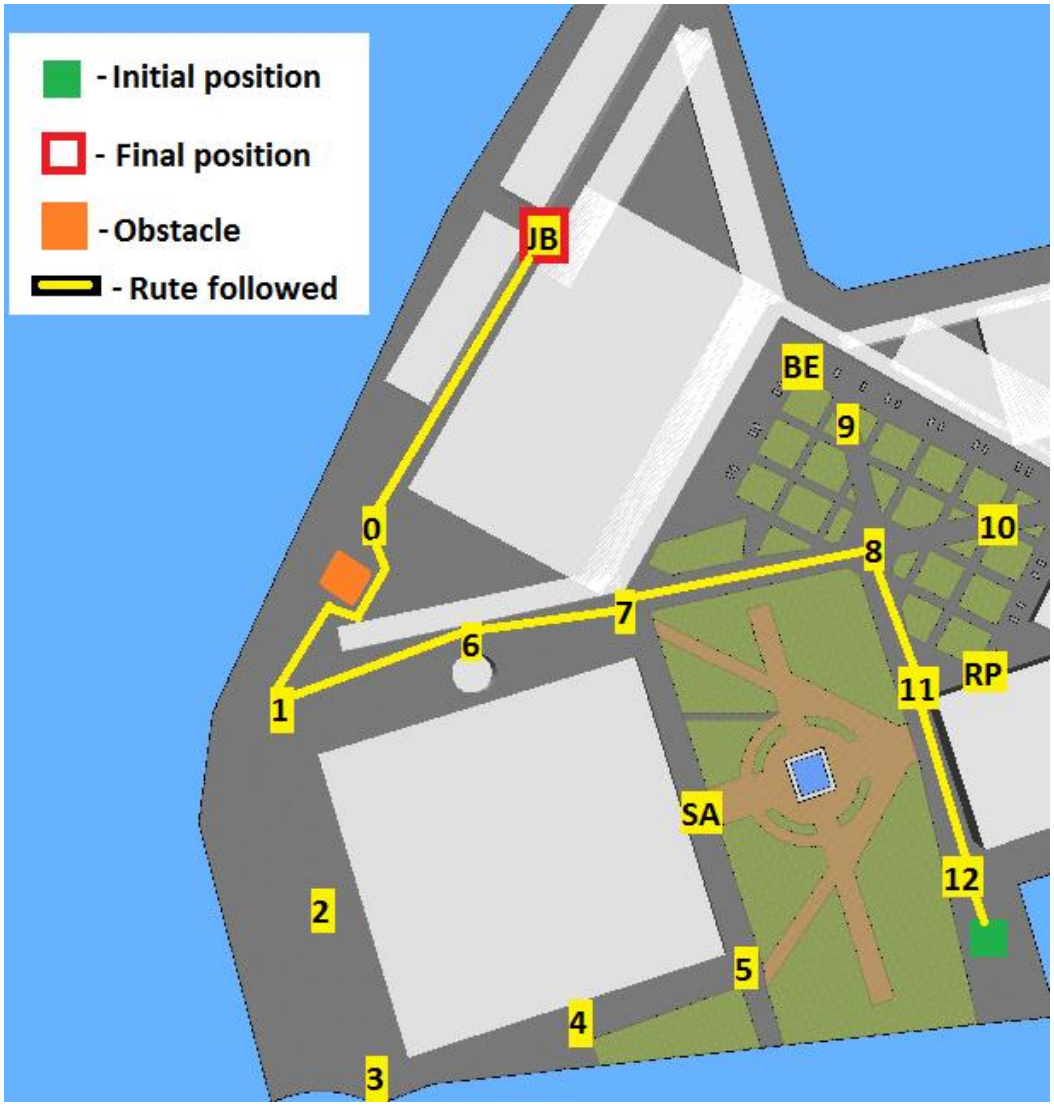


Figura 126: Path followed by the autonomous vehicle in Scenario 2

The autonomous vehicle travels the route established by the pathfinding algorithm correctly. In the course from waypoint 1 to waypoint 0 the autonomous vehicle prevents the collision with the introduced obstacle and goes to its target. The result of the experiments in this scenario it is satisfactory.